

2

## Tipos de datos, variables y expresiones

Grado en Ingeniería Electrónica de Comunicaciones

Raquel Hervás Ballesteros

Luis Hernández Yáñez

Virginia Francisco Gilmartín

Javier Arroyo Gallardo

Facultad de Informática  
Universidad Complutense



# Índice

---

Sintaxis / Semántica

Un ejemplo de programación

Primer programa en C++

Salida por pantalla

Cálculos

Variables

Entrada por teclado

Resolución de problemas

Operadores

Constantes

Formato de salida



Informática: Tipos de datos, variables e instrucciones



# Sintaxis vs Semántica

---

## Sintaxis

- Reglas que determinan cómo se pueden construir y secuenciar los elementos del lenguaje

## Semántica

- Significado de cada elemento del lenguaje ¿Para qué sirve?

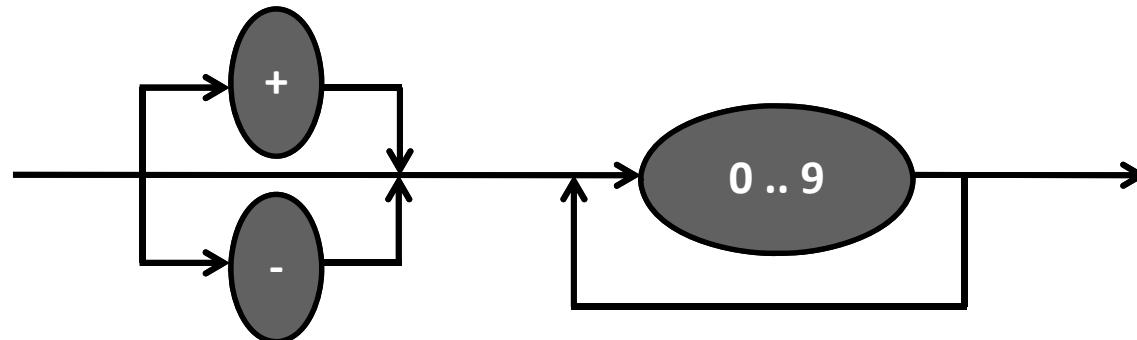


# Sintaxis de los lenguajes de programación

## Especificación

- ✓ Lenguajes (BNF)
- ✓ Diagramas

Ejemplo: Números enteros (sin decimales)

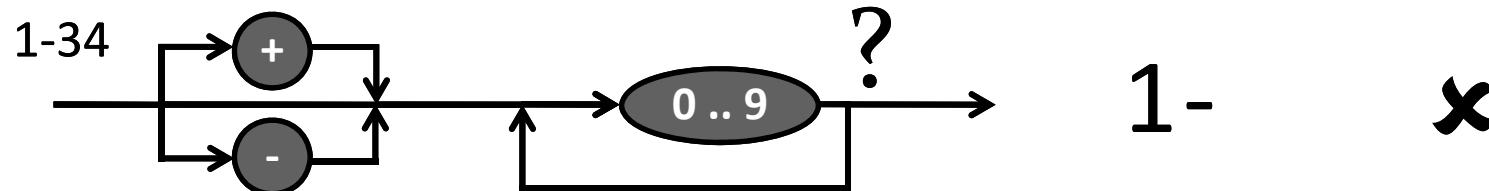
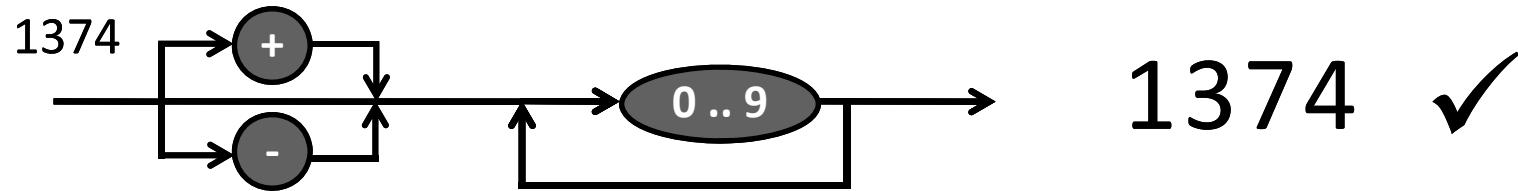
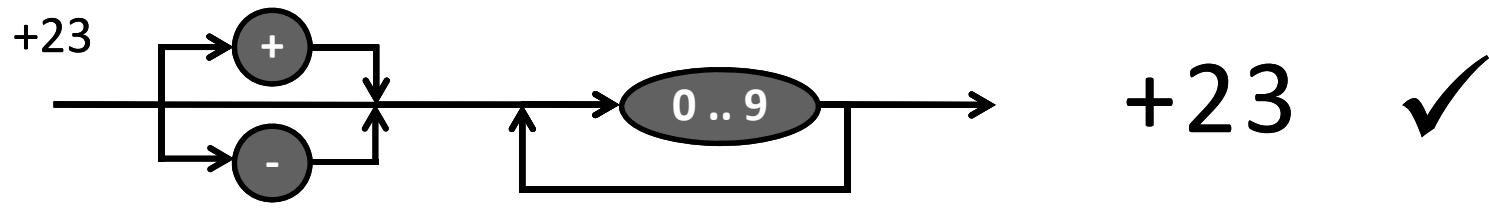


+23	✓
-159	✓
1374	✓
1-34	✗
3.4	✗
002	✓



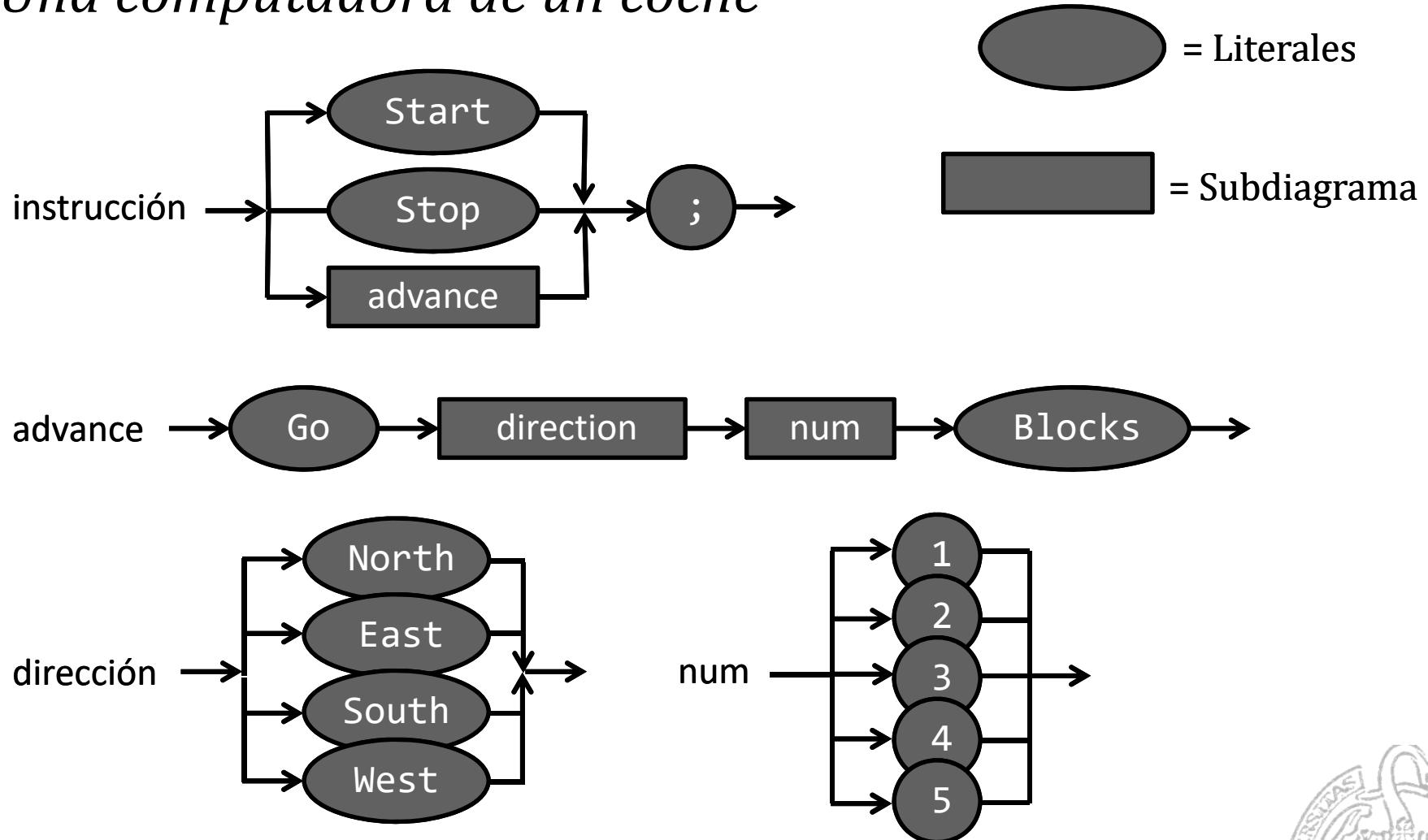
# Diagramas de sintaxis

---



# Ejemplo de programación (I)

*Una computadora de un coche*



# Ejemplo de programación (II)

*El problema a resolver*

*Estando el coche en la posición A, conseguir llegar al Cine Tívoli (B)*



*¿Qué pasos hay que seguir?*

*Arrancar*

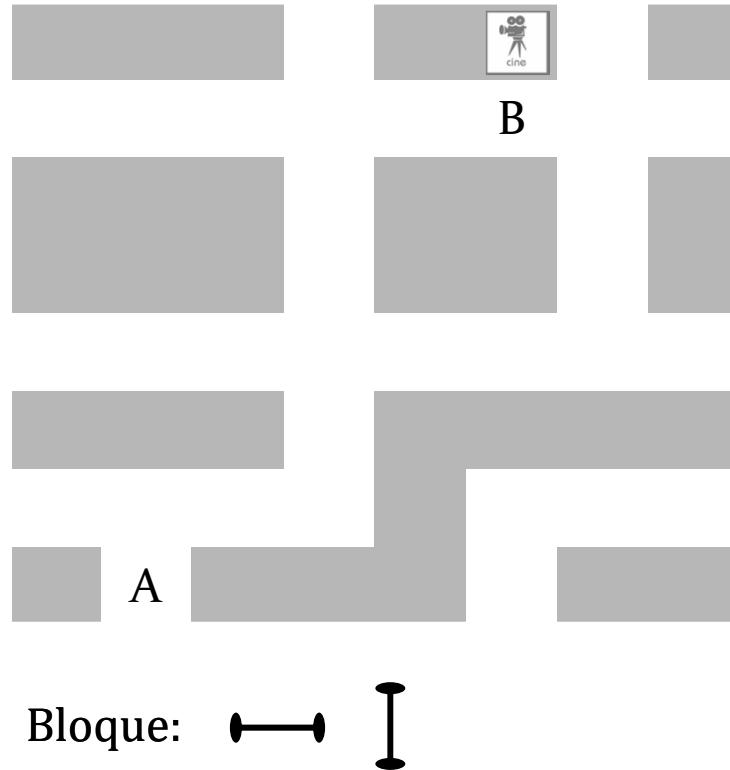
*Ir un bloque al Norte*

*Ir dos bloques al Este*

*Ir cinco bloques al Norte*

*Ir dos bloques al Este*

*Parar*



# Ejemplo de programación (III)

## *El algoritmo*

Secuencia de pasos que hay que seguir para resolver el problema

1.- *Arrancar*

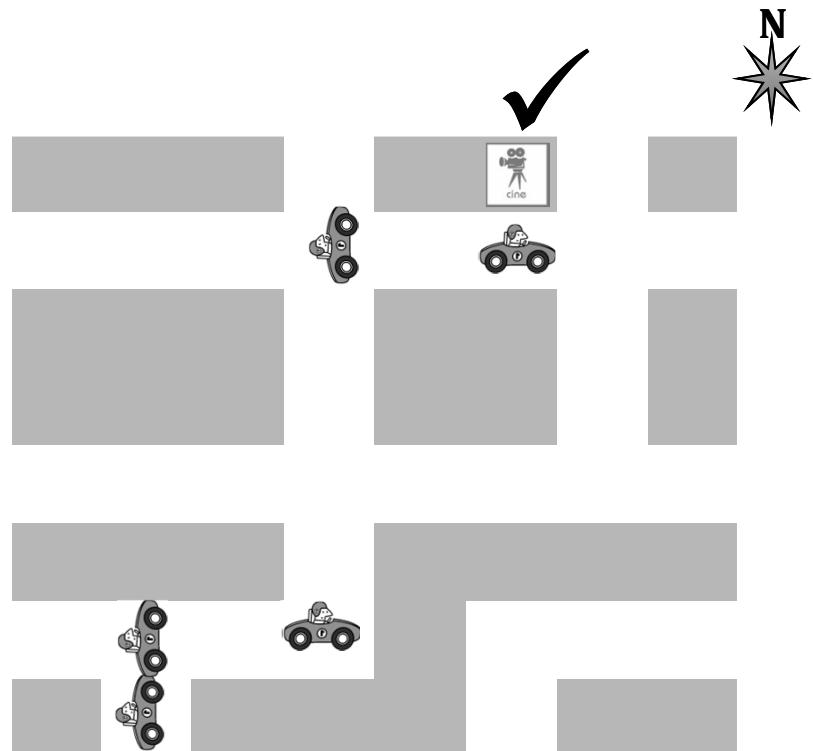
2.- *Ir un bloque al Norte*

3.- *Ir dos bloques al Este*

4.- *Ir cinco bloques al Norte*

5.- *Ir dos bloques al Este*

6.- *Parar*



Esos pasos sirven tanto para  
una persona como para una computadora.

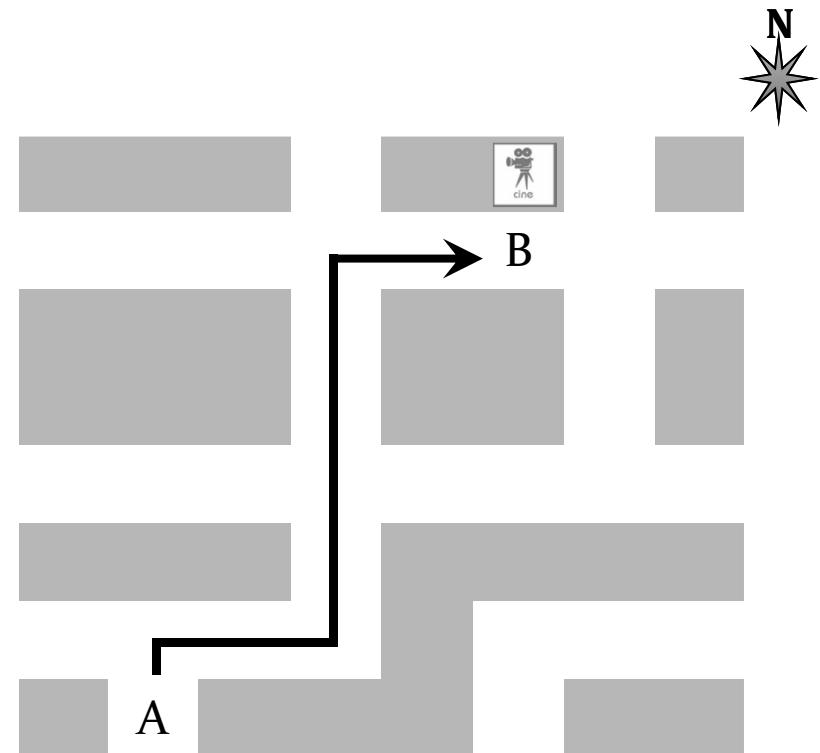


# Ejemplo de programación (IV)

## *El programa*

InSTRUCCIONES escritas en  
el lenguaje de programación

```
Start;  
Go North 1 Blocks;  
Go East 2 Blocks;  
Go North 5 Blocks;  
Go East 2 Blocks;  
Stop;
```



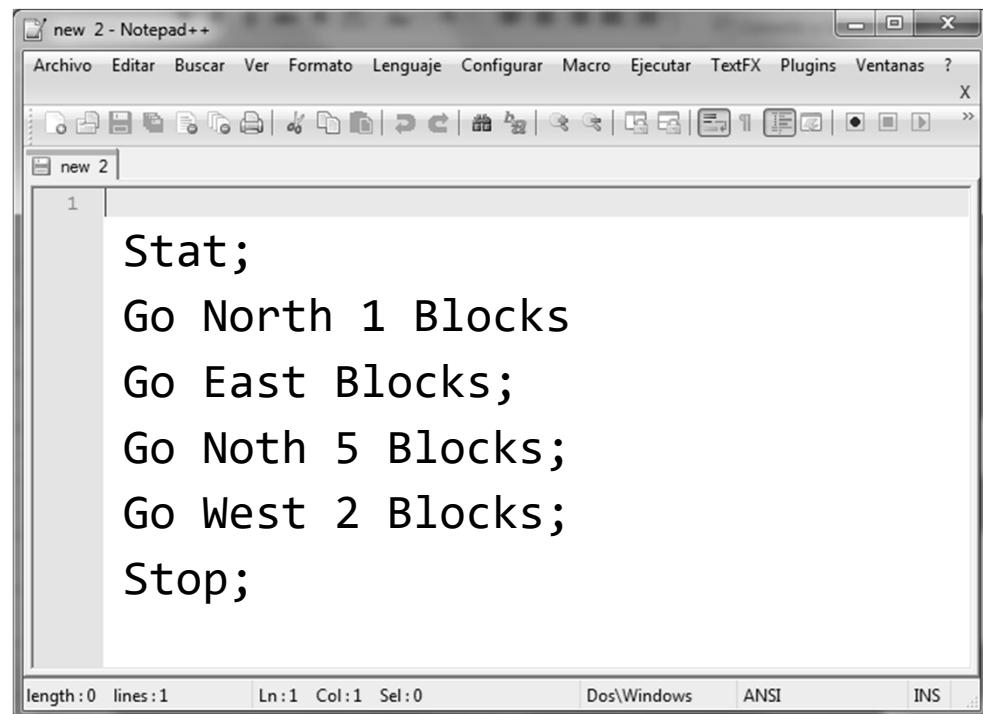
- 1.- Arrancar
- 2.- Ir un bloque al Norte
- 3.- Ir dos bloques al Este
- 4.- Ir cinco bloques al Norte
- 5.- Ir dos bloques al Este
- 6.- Parar



# Ejemplo de programación (V)

---

Escribimos el código del programa en un editor y lo guardamos en un archivo:



A screenshot of the Notepad++ text editor window. The title bar says "new 2 - Notepad++". The menu bar includes Archivo, Editar, Buscar, Ver, Formato, Lenguaje, Configurar, Macro, Ejecutar, TextFX, Plugins, Ventanas, and ?.

The toolbar below the menu has icons for file operations like Open, Save, Print, and Find.

The main text area contains the following program code:

```
1
Stat;
Go North 1 Blocks
Go East Blocks;
Go Noth 5 Blocks;
Go West 2 Blocks;
Stop;
```

The status bar at the bottom shows "length : 0 lines : 1 Ln :1 Col :1 Sel :0 Dos\Windows ANSI INS".

Copiamos el archivo  
en una llave USB  
y lo llevamos al coche

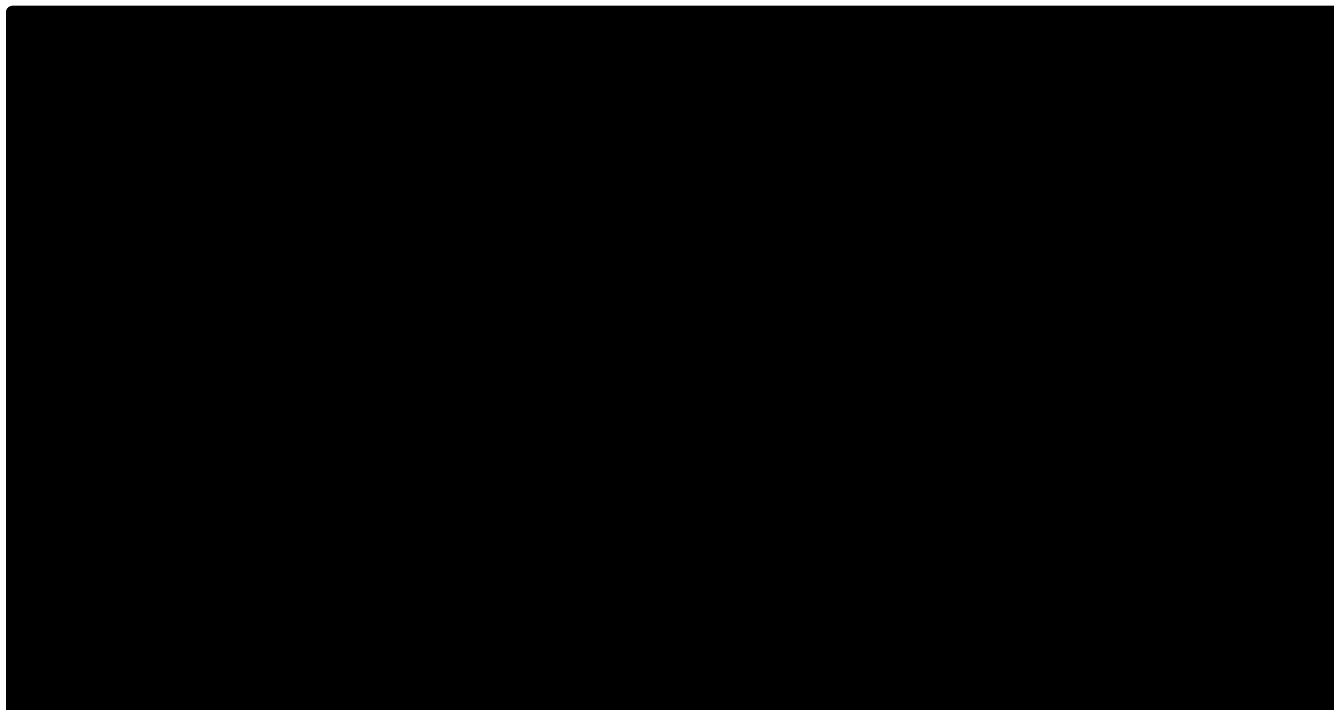


# Ejemplo de programación (VI)

---

## *La compilación*

Introducimos la llave USB en el coche  
y pulsamos el botón de ejecutar el programa:



Errores  
de sintaxis



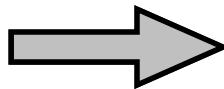
# Ejemplo de programación (VII)

---

## *Depuración*

Editamos el código para corregir los errores sintácticos:

```
Stat;  
Go North 1 Blocks  
Go East Blocks;  
Go Noth 5 Blocks;  
Go West 2 Blocks;  
Stop;
```



```
Start;  
Go North 1 Blocks;  
Go East 3 Blocks;  
Go North 5 Blocks;  
Go West 2 Blocks;  
Stop;
```



# Ejemplo de programación (VIII)

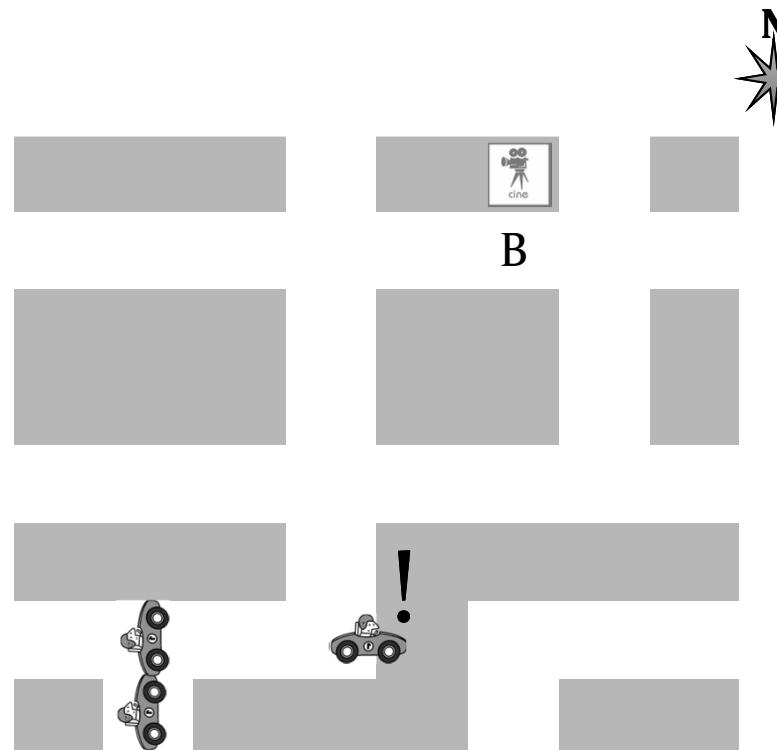
## *La ejecución*

Se realiza lo que pide cada instrucción:

Start;

Go North 1 Blocks;

Go East 3 Blocks;



Error de ejecución

*¡Una instrucción no se puede ejecutar!*



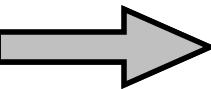
# Ejemplo de programación (IX)

---

## *Depuración*

Editamos el código para arreglar el error de ejecución:

```
Start;  
Go North 1 Blocks;  
Go East 3 Blocks;  
Go North 5 Blocks;  
Go West 2 Blocks;  
Stop;
```



```
Start;  
Go North 1 Blocks;  
Go East 2 Blocks;  
Go North 5 Blocks;  
Go West 2 Blocks;  
Stop;
```



# Ejemplo de programación (X)

## *La ejecución*

Se realiza lo que pide cada instrucción:

Start;

Go North 1 Blocks;

Go East 2 Blocks;

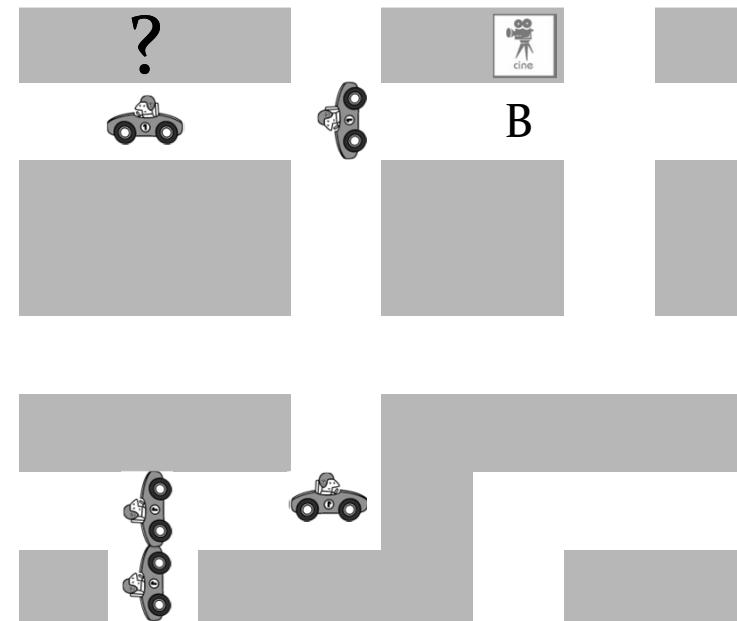
Go North 5 Blocks;

Go West 2 Blocks;

Stop;

Error lógico

*¡El programa no llega al resultado deseado!*



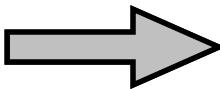
# Ejemplo de programación (XI)

---

## *Depuración*

Editamos el código para arreglar el error lógico:

```
Start;  
Go North 1 Blocks;  
Go East 2 Blocks;  
Go North 5 Blocks;  
Go West 2 Blocks;  
Stop;
```



```
Start;  
Go North 1 Blocks;  
Go East 2 Blocks;  
Go North 5 Blocks;  
Go East 2 Blocks;  
Stop;
```



# Ejemplo de programación (XII)

## *La ejecución*

Se realiza lo que pide cada instrucción:

Start;

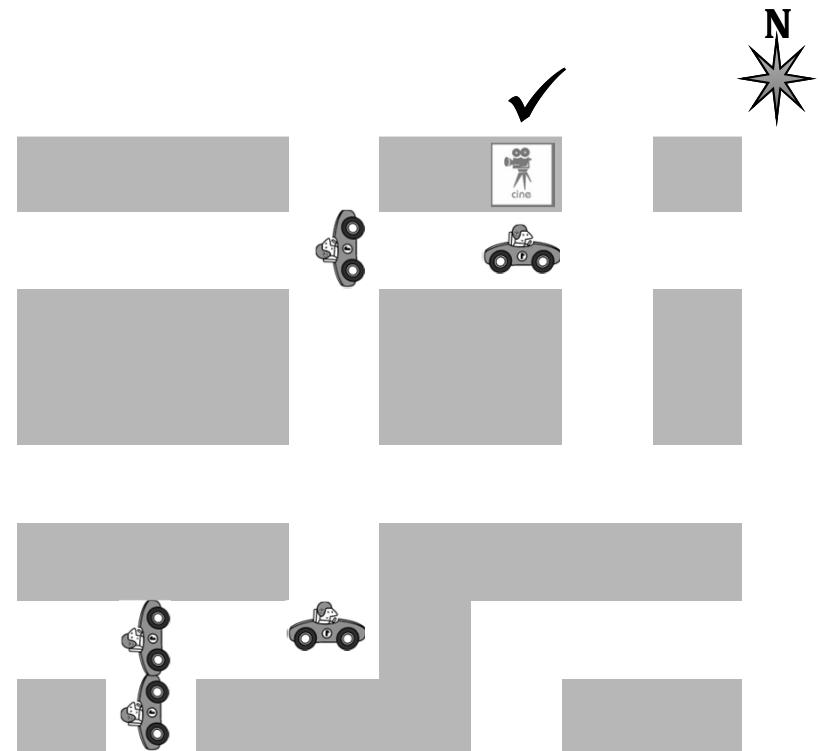
Go North 1 Blocks;

Go East 2 Blocks;

Go North 5 Blocks;

Go East 2 Blocks;

Stop;



*¡Conseguido!*



# El programa principal (I)

---

La función `main()`: *donde comienza la ejecución...*

```
#include <iostream>
using namespace std;

int main()    // main() es donde empieza la ejecución
{
    cout << "Hola Mundo!" << endl;
    return 0;
}
```

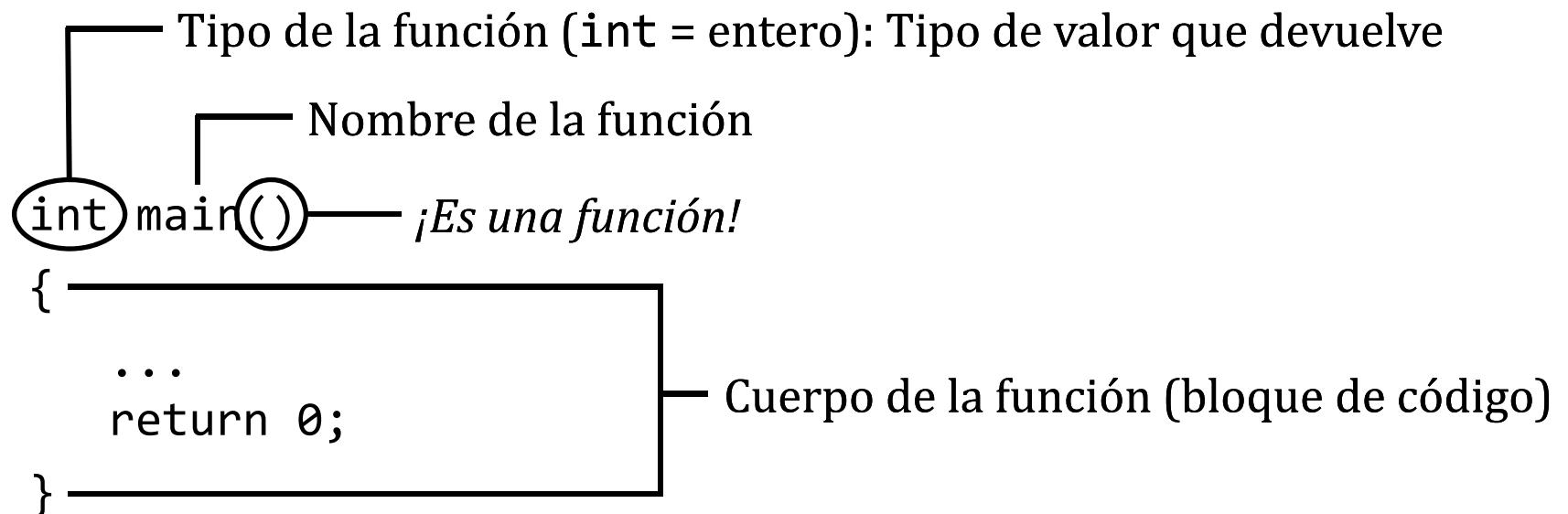
Contiene las instrucciones que hay que ejecutar



# El programa principal (II)

---

La función `main()`:



`return 0;`

Devuelve el resultado (0) de la función



# Documentación del código

---

Comentarios (se ignoran):

```
#include <iostream>
using namespace std;
```

```
int main() // main() es donde empieza la ejecución
{
    cout << "Hola Mundo!" << endl;
    ...
}
```

Hasta el final de la línea: // Comentario de una línea  
De varias líneas: /\* Comentario de varias  
líneas seguidas \*/



# La infraestructura

---

Código para reutilizar:

```
#include <iostream> ← Una directiva: empieza por #
using namespace std;
```

```
int main() // main() es donde empieza la ejecución
{
    cout << "Hola Mundo!" << endl;
    return 0;
}
```

Bibliotecas de funciones a nuestra disposición



# Bibliotecas y espacios de nombres

---

## *Bibliotecas*

Se incluyen con la *directiva #include*:

```
#include <iostream>
```

(Utilidades de entrada/salida por consola)

Para mostrar o leer datos hay que incluir la biblioteca **iostream**

## *Espacios de nombres*

En **iostream** hay espacios de nombres; ¿cuál queremos?

```
#include <iostream>
using namespace std; ← Es una instrucción: termina en ;
```

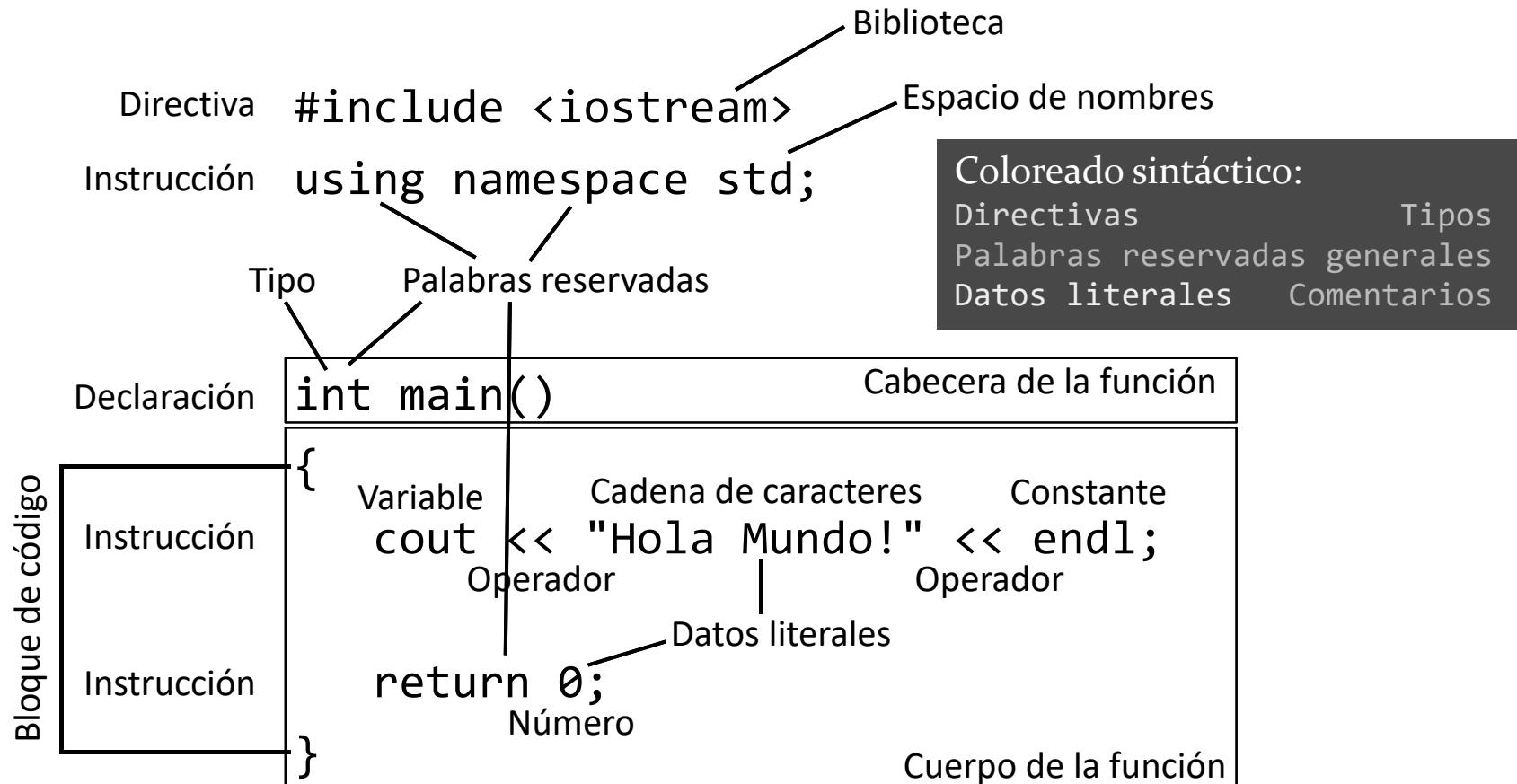
Siempre usaremos el espacio de nombres estándar (**std**)

Muchas bibliotecas no tienen espacios de nombres



# Elementos del programa

## *Elementos del programa*



Las instrucciones terminan en ;



# Espacios en blanco

---

Para una mayor legibilidad debéis separar los elementos por uno o más *espacios en blanco* (espacios, tabuladores y saltos de línea)

El compilador los ignorará pero hará el código mucho más fácil de leer

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Hola Mundo!" << endl;
    return 0;
}
```

```
#include <iostream> using namespace std;
int main(){cout<<"Hola Mundo!"<<endl;
return 0;}
```

¿Cuál se lee mejor?



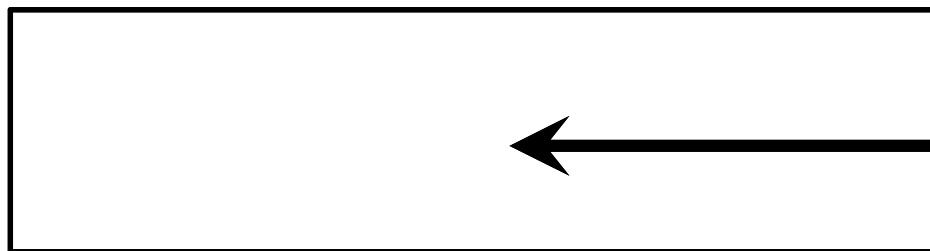
# Programa mínimo

---

Una plantilla para empezar:

```
#include <iostream>
using namespace std;
```

```
int main()
{
```



*¡Tu código aquí!*

```
    return 0;
```

```
}
```



# ***El Quijote...***

---

*... recitado en la consola*



Mostrar los textos con cout <<:

```
#include <iostream>
using namespace std;

int main()
{
    cout << "En un lugar de la Mancha," << endl;
    cout << "de cuyo nombre no quiero acordarme," << endl;
    cout << "no ha mucho tiempo que vivía un hidalgo de los de
lanza en astillero, ..." << endl;
    return 0;
}
```



# Introducción del código del programa (I)

---

Terminamos cada línea de código con un salto de línea (↓):

```
#include <iostream> ↓  
using namespace std; ↓  
↓  
int main() ↓  
{ ↓  
    cout << "En un lugar de la Mancha," << endl; ↓  
    cout << "de cuyo nombre no quiero acordarme," << endl; ↓  
    cout << "no ha mucho tiempo que vivía un hidalgo de los de  
lanza en astillero, ..." << endl; ↓  
    return 0; ↓  
} ↓
```



# Introducción del código del programa (II)

---

No hay que partir una cadena literal entre dos líneas:

```
cout << "no ha mucho tiempo que vivía un hidalgo de ↵  
los de lanza en astillero, ..." << endl; ↴
```

*¡La cadena no termina (1<sup>a</sup> línea)!*

*¡No se entiende los (2<sup>a</sup> línea)!*



# Mantenimiento y reusabilidad

---

*Hay que programar pensando en posibles cambios*

- ✓ Usa espacio en blanco para separar los elementos:

```
cout << "En un lugar de la Mancha," << endl;  
      △ △           △ △
```

mejor que

```
cout<<"En un lugar de la Mancha,"<<endl;
```

- ✓ Usa sangría (indentación) para el código de un bloque:

```
{  
Tab → cout << "En un lugar de la Mancha," << endl;  
ó   | ...  
3 esp. | return 0;  
}
```

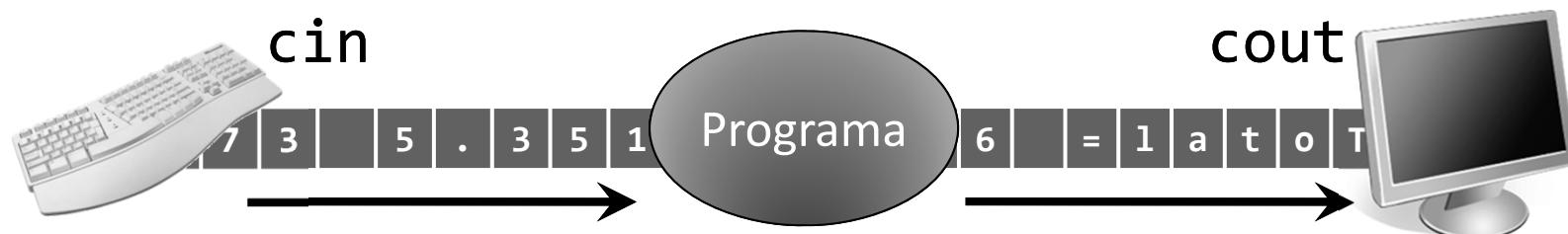
*¡El estilo importa!*



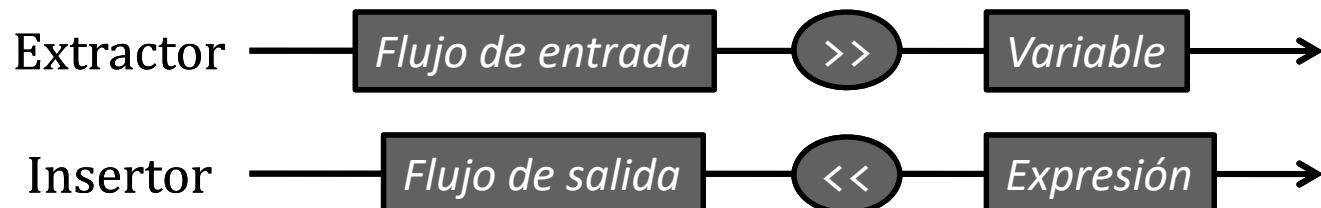
# Entrada/salida por consola (teclado/pantalla)

## Flujos de texto (streams)

- ✓ Conectan la ejecución del programa con los dispositivos de E/S
- ✓ Son secuencias de caracteres
- ✓ Entrada por teclado: flujo de entrada `cin` (tipo `istream`)
- ✓ Salida por pantalla: flujo de salida `cout` (tipo `ostream`)



Biblioteca `iostream` con espacio de nombres `std`



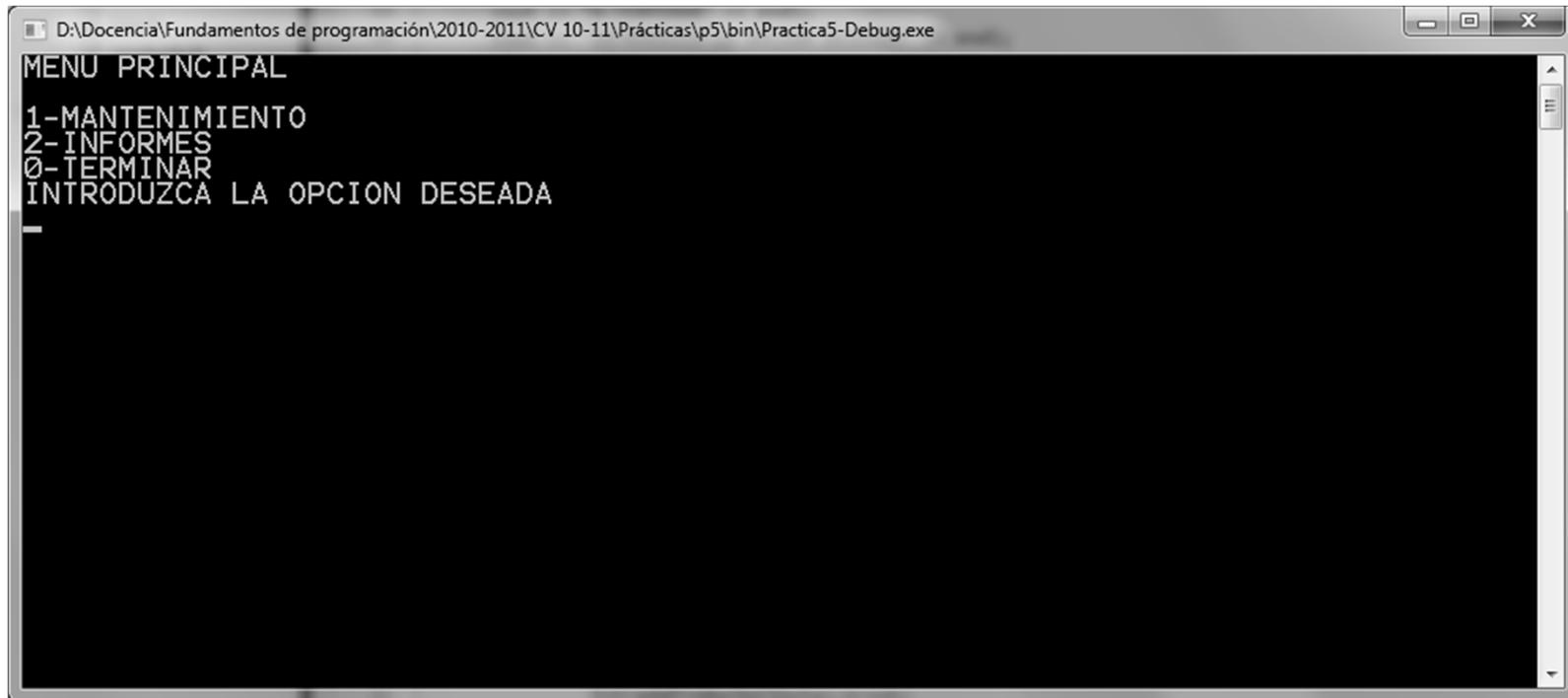
# El dispositivo de salida

---

*Pantalla en modo texto*

→ Líneas de 80 caracteres (textos)

Aplicación en modo texto



# Visualización de datos (I)

*El insertor <<*

`cout << ...;`

Inserta textos en la pantalla a partir de la posición del cursor

*Line wrap* (continúa en la siguiente línea si no cabe)

Se pueden encadenar:

`cout << ... << ... << ...;`

Recuerda: las instrucciones terminan en ;



# Visualización de datos (II)

---

*Con el insertor << podemos mostrar...*

- ✓ Cadenas de caracteres literales: Textos encerrados entre comillas dobles: "..."

`cout << "Hola Mundo!" ;`

*¡Las comillas no se muestran!*

- ✓ Números literales con o sin decimales, con signo o no (123, -37, 3.1416, ...)

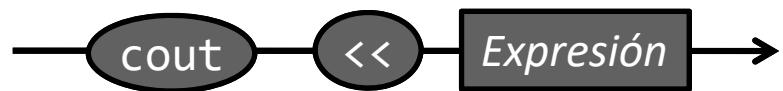
`cout << "Pi = " << 3.1416;` *¡Punto decimal, NO coma!*

Se muestran los caracteres que representan el número

- ✓ endl: Salto de línea



# Salida por pantalla



```
int meses = 7;  
cout << "Total: " << 123.45 << endl << "Meses: " << meses;  
cout << 123.45 << endl << "Meses: " << meses;  
cout << endl << "Meses: " << meses;  
cout << "Meses: " << meses;  
cout << meses;
```

The code demonstrates the use of cout for output. It first outputs "Total: " followed by the value 123.45 and a new line. Then it outputs "Meses: " followed by the variable meses. The output is visualized as text on a screen, with curly braces underlining the segments of the cout statements to show how they correspond to the displayed text.



# Cálculos en los programas (I)

---

## *Operadores aritméticos*

- + Suma
- Resta
- \* Multiplicación
- / División

Operadores binarios

*operando\_izquierdo      operador      operando\_derecho*

Operación	Resultado
3 + 4	7
2.56 - 3	-0.44
143 * 2	286
45.45 / 3	15.15



# Cálculos en los programas (II)

---

## *Números literales (concretos)*

- ✓ Enteros: sin parte decimal

Signo negativo (opcional) + secuencia de dígitos

3      143      -12      67321      -1234

No se usan puntos de millares

- ✓ Reales: con parte decimal

Signo negativo (opcional) + secuencia de dígitos  
+ punto decimal + secuencia de dígitos

3.1416    357.0    -1.333    2345.6789    -404.1



Punto decimal (3.1416), NO coma (3,~~1416~~)



# Cálculos en los programas. Ejemplo (I)

cálculos.cpp

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Un texto" << endl;
    cout << "Un número" << endl;

    cout << "133 + 1234 = " << 133 + 1234 << endl;
    cout << "1234 - 111.5 = " << 1234 - 111.5 << endl;
    cout << "34 * 59 = " << 34 * 59 << endl;
    cout << "3.4 * 5.93 = " << 3.4 * 5.93 << endl;
    cout << "500 / 3 = " << 500 / 3 << endl; // Div. entera
    cout << "500.0 / 3 = " << 500.0 / 3 << endl; // Div. real

    return 0;
}
```



# Cálculos en los programas. Ejemplo (II)

```
D:\FP\Tema02>g++ -o cálculos cálculos.cpp
Información: se resuelve std::cout al enlazamiento)
c:/mingw/bin/../../lib/gcc/mingw32/4.5.0/../../..
importación automática se activó sin especificar de órdenes.
Esto debe funcionar a menos que involucre esterencien símbolos de DLLs auto-importadas.

D:\FP\Tema02>cálculos
133 + 1234 = 1367
1234 - 111.5 = 1122.5
34 * 59 = 2006
3.4 * 5.93 = 20.162
500 / 3 = 166
500.0 / 3 = 166.667
```

División entera

División real



# División entera vs división real

---

Ambos operandos enteros → División entera

Algún operando real → División real

División	Resultado
$500 / 3$	166
$500.0 / 3$	166.667
$500 / 3.0$	166.667
$500.0 / 3.0$	166.667

Comprueba siempre que el tipo de división sea el que quieras



# Variables. Definición

---

Espacio de memoria que contiene un dato de cierto tipo que puede variar.

- En un trocito de memoria guardamos un número, un carácter, un texto...

Las variables son datos a los que se accede por medio de un nombre

Las variables deben de ser declaradas indicando el tipo del dato que almacenan y el nombre con el que nos vamos a referir a ellas



# Variables. Declaración (I)

---

*tipo nombre;*

int cantidad;

double precio;

Se reserva espacio suficiente

Memoria	
cantidad	?
precio	?

...

LAS VARIABLES NO SE INICIALIZAN AUTOMÁTICAMENTE

No se deben usar hasta que se les haya dado algún valor

*¿Dónde colocamos las declaraciones?*

Siempre antes del primer uso

Habitualmente al principio de la función



# Variables. Declaración (II)

```
#include <iostream>
using namespace std;

int main()
{
    int cantidad;
    double precio, total;

    return 0;
}
```

Memoria	
cantidad	?
precio	?
total	?
	...

Podemos declarar varias de un mismo tipo separando los nombres con comas



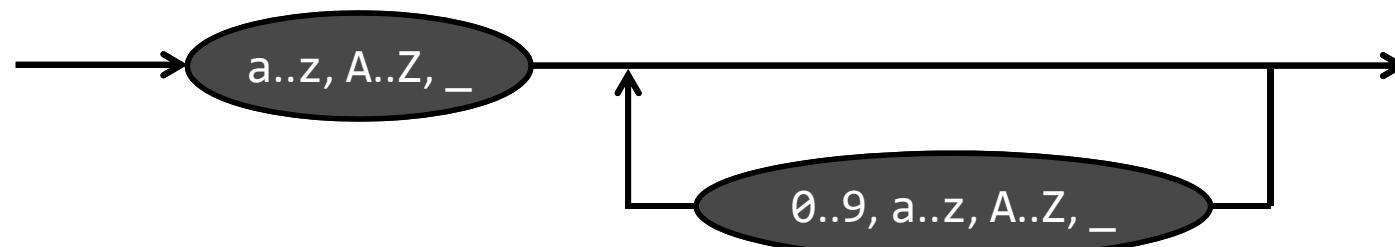
# Identificadores (I)

≠ palabras reservadas

Nombres para variables que sirven para accederlas / modificarlas

Deben ser descriptivos

Sintaxis:



cantidad    precio    total    base    altura    area    \_numerador



*¡Ni eñes ni vocales acentuadas!*



# Palabras reservadas del lenguaje C++

---

```
asm  auto  bool  break  case  catch  char  class  const  
const_cast  continue  default  delete  do  double  
dynamic_cast  else  enum  explicit  extern  false  
float  for  friend  goto  if  inline  int  long  
mutable  namespace  new  operator  private  protected  
public  register  reinterpret_cast  return  short  
signed  sizeof  static  static_cast  struct  switch  
template  this  throw  true  try  typedef  typeid  
typename  union  unsigned  using  virtual  void  
volatile  while
```



# Mayúsculas y minúsculas

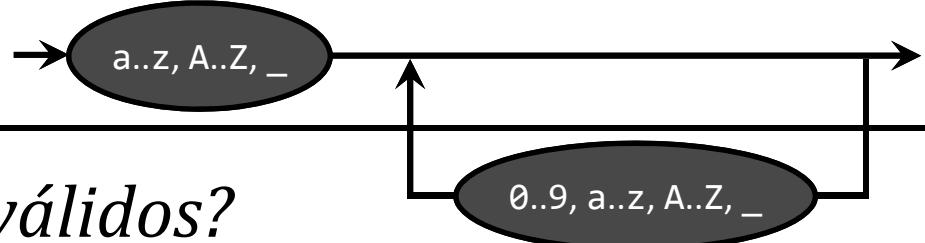
---

C++ distingue entre mayúsculas y minúsculas. Ejemplos:

- `int` es una palabra reservada de C++ para declarar datos enteros, pero `Int`, `INT` o `inT` no son palabras reservadas de C++
- `true` es una palabra reservada de C++ para el valor *verdadero*, pero `True` o `TRUE` no son palabras reservadas de C++



# Identificadores (II)



¿Qué identificadores son válidos?

balance ✓

interesAnual ✓

\_base\_imponible ✓

años ✗

EDAD12 ✓

salario\_1\_mes ✓

\_edad ✓

cálculoNómina ✗

valor%100 ✗

AlgunValor ✓

100caracteres ✗

valor? ✗

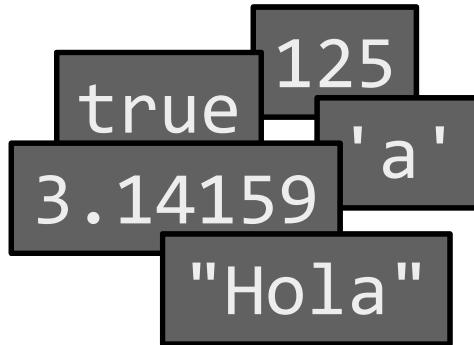
\_12\_meses ✓

\_\_\_\_\_valor ✓



# Tipos de datos

---



Cada dato, de un tipo concreto

Cada tipo establece:

- El conjunto (intervalo) de valores válidos
- El conjunto de operaciones que se pueden realizar



# Tipos de datos básicos

---

**int**

Números enteros (sin decimales) 1363, -12, 49

**float**

Números reales 12.45, -3.1932, 1.16E+02

**double**

Números reales (mayor intervalo y precisión)

**char**

Caracteres 'a', '{', '\t'

**bool**

Valores lógicos (verdadero/falso) true, false

**string**

Cadenas de caracteres (biblioteca string) "Hola Mundo!"

**void**

*Nada, ausencia de tipo, ausencia de dato (funciones)*



**char**

## *Caracteres*

## Intervalo de valores: Juego de caracteres (ASCII)

1 byte

### ✓ Literales:

'a'      '%'      '1'

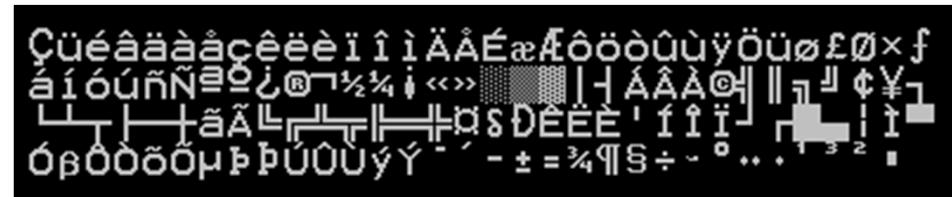
✓ Constantes de barra invertida (o secuencias de escape)

### ➤ Caracteres de control:

'\t' = tabulador    '\n' = salto de línea ...

! "#\$%&' ()\*+, - . /  
0123456789: ; <=>?  
@ABCDEFGHIJKLMNO  
PQRSTUVWXYZ[\]^\_  
' abcdefghijklmno  
pqrstuvwxyz{|}~

## ASCII (códigos 32..127)



## ISO-8859-1



# bool

## *Valores lógicos*

Sólo dos valores posibles:

- Verdadero (*true*)
- Falso (*false*)

Literales:

`true`    `false`

El `0` es equivalente a `false`

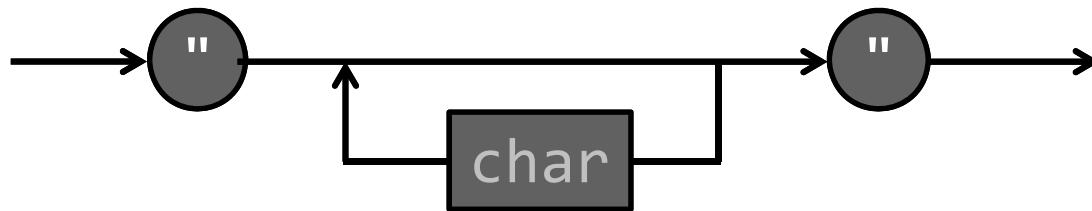
Cualquier número distinto de `0` es equivalente a `true`



# string

## Cadenas de caracteres

"Hola" "Introduce el numerador: " "X142FG5TX?%A"



Secuencias de caracteres

Programas con variables de tipo **string**:

```
#include <string>
using namespace std;
```



Las comillas tipográficas (apertura/cierre) ` o ' duplicadas  
NO sirven. Asegúrate de utilizar comillas rectas: "..."



# Tipos de datos básicos: ejemplo

tipos.cpp

```
#include <iostream>
#include <string>
using namespace std; // Un solo using... para ambas bibliotecas

int main()
{
    int entero = 3; // Podemos asignar (inicializar) al declarar
    double real = 2.153;
    char caracter = 'a';
    bool cierto = true;
    string cadena = "Hola";
    cout << "Entero: " << entero << endl;
    cout << "Real: " << real << endl;
    cout << "Carácter: " << caracter << endl;
    cout << "Booleano: " << cierto << endl;
    cout << "Cadena: " << cadena << endl;

    return 0;
}
```

D:\FP\Tema2>tipos  
Entero: 3  
Real: 2.153  
Carácter: a  
Booleano: 1  
Cadena: Hola  
D:\FP\Tema2>



# Modificadores de tipos

---

- signed / unsigned : con signo (por defecto) / sin signo
- short / long : menor / mayor intervalo de valores

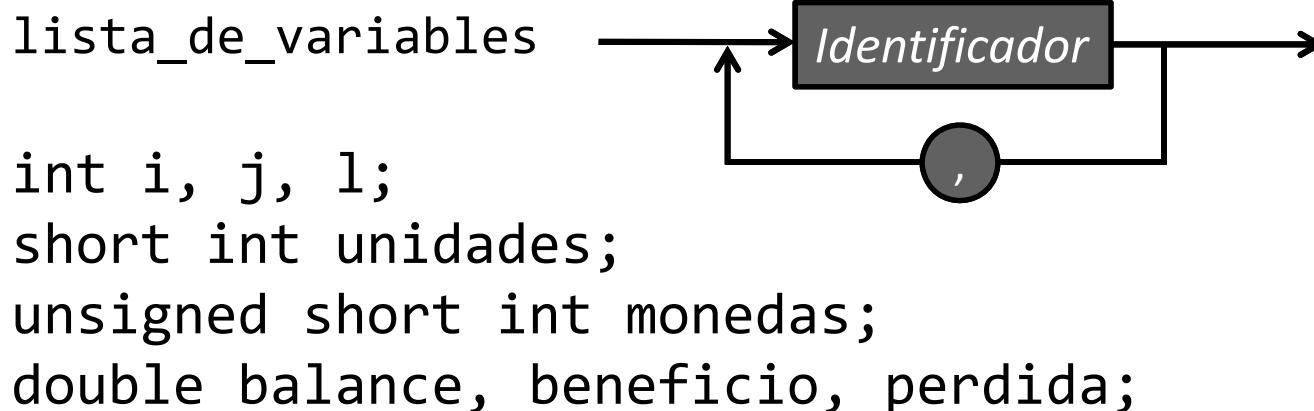
Tipo	Intervalo	
int	-2147483648 .. 2147483647	
unsigned int	0 .. 4294967295	4 bytes
short int	-32768 .. 32768	2 bytes
unsigned short int	0 .. 65535	
long int	-2147483648 .. 2147483647	4 bytes
unsigned long int	0 .. 4294967295	
double	+ - 2.23e-308 .. 1.79e+308	8 bytes
long double	+ - 3.37E-4932 .. 1.18E+4932	12 bytes



# Declaración de variables

---

```
[ modificadores] tipo lista_de_variables;  
└────────── Opcional ──────────┘
```



*Programación con buen estilo:*

Identificadores descriptivos

Espacio tras cada coma

Nombres de las variables en minúsculas

(Varias palabras: capitaliza cada inicial: `interesPorMes`)



# Inicialización de variables

---

¡En C++ las variables no se inicializan automáticamente!

*¡Una variable debe ser haber sido inicializada antes de ser accedida!*

¿Cómo se inicializa una variable?

- Al asignarle un valor (instrucción de asignación)
- Al declararla
- Al leer su valor de teclado (`cin >>`)

Inicialización en la propia declaración:



```
int i = 0, j, l = 26;  
short int unidades = 100;
```

En particular, una expresión  
puede ser un literal



# Uso de las variables

---

## *Obtención del valor de una variable*

- ✓ Nombre de la variable en una expresión

```
cout << balance;  
interesPorMes * meses / 100;
```

## *Modificación del valor de una variable*

- ✓ Nombre de la variable a la izquierda del =

```
balance = 1214;  
porcentaje = valor / 30;
```

Las variables han de haber sido previamente declaradas



# Variables. Asignación de valores

---

*operador =*

*variable = expresión;* ← Instrucción: termina en ;

El valor asignado a una variable puede ser:

- Un literal (valor concreto)

edad = 19; // variable edad y literal 19

- Una expresión

a = b + c; // variables a, b y c

```
cantidad = 12; // int  
precio = 39.95; // double  
total = cantidad * precio; // Asigna 479.4
```

cantidad ← 12

Concordancia de tipos:

~~cantidad = 12.5;~~

*¡¡¡A la izquierda del = debe ir siempre una variable!!!*



# Instrucciones de asignación. Errores

---

```
int a, b, c;
```

~~5 = a;~~

// ERROR: un literal no puede recibir un valor

~~a + 23 = 5;~~

// ERROR: no puede haber una expresión a la izda.

~~b = "abc";~~

// ERROR: un entero no puede guardar una cadena

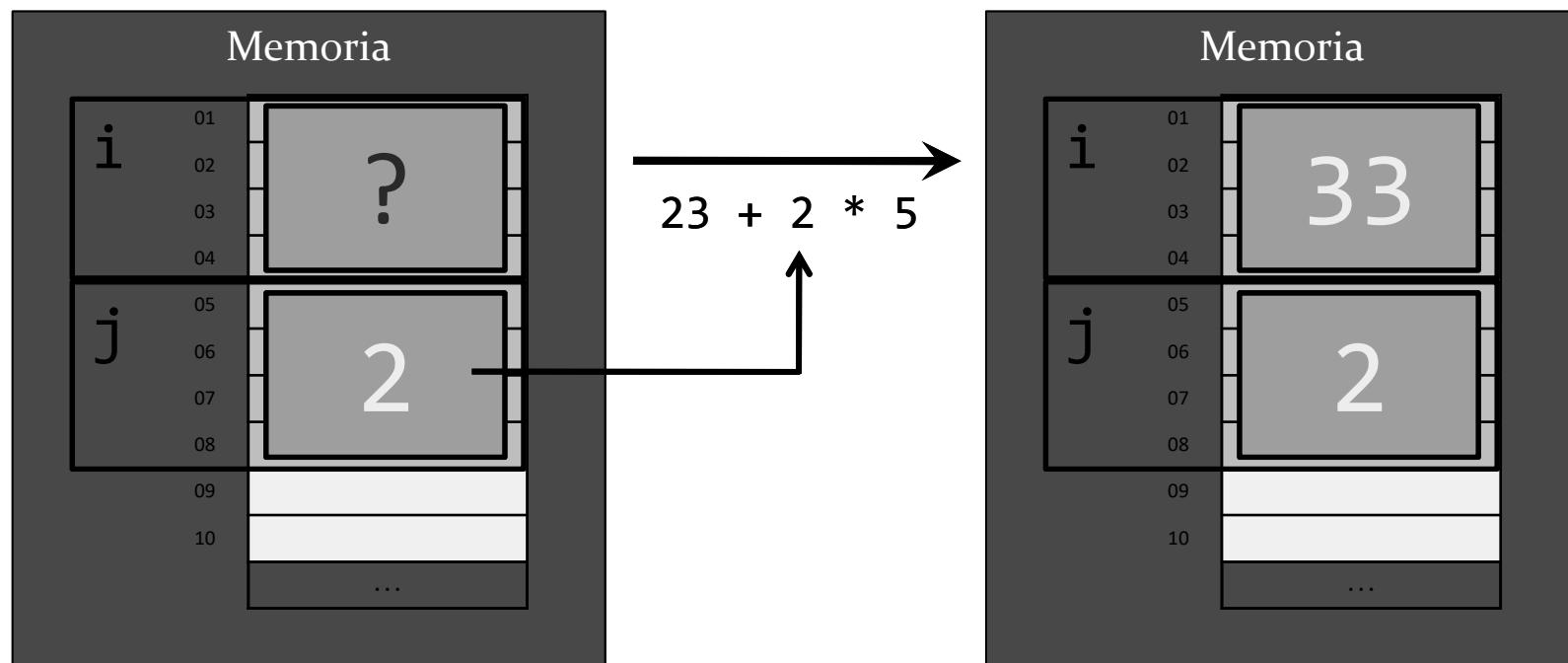
~~c = 23 5;~~

// ERROR: expresión no válida (falta operador)



# Variables, asignación y memoria

```
int i, j = 2;  
i = 23 + j * 5;
```

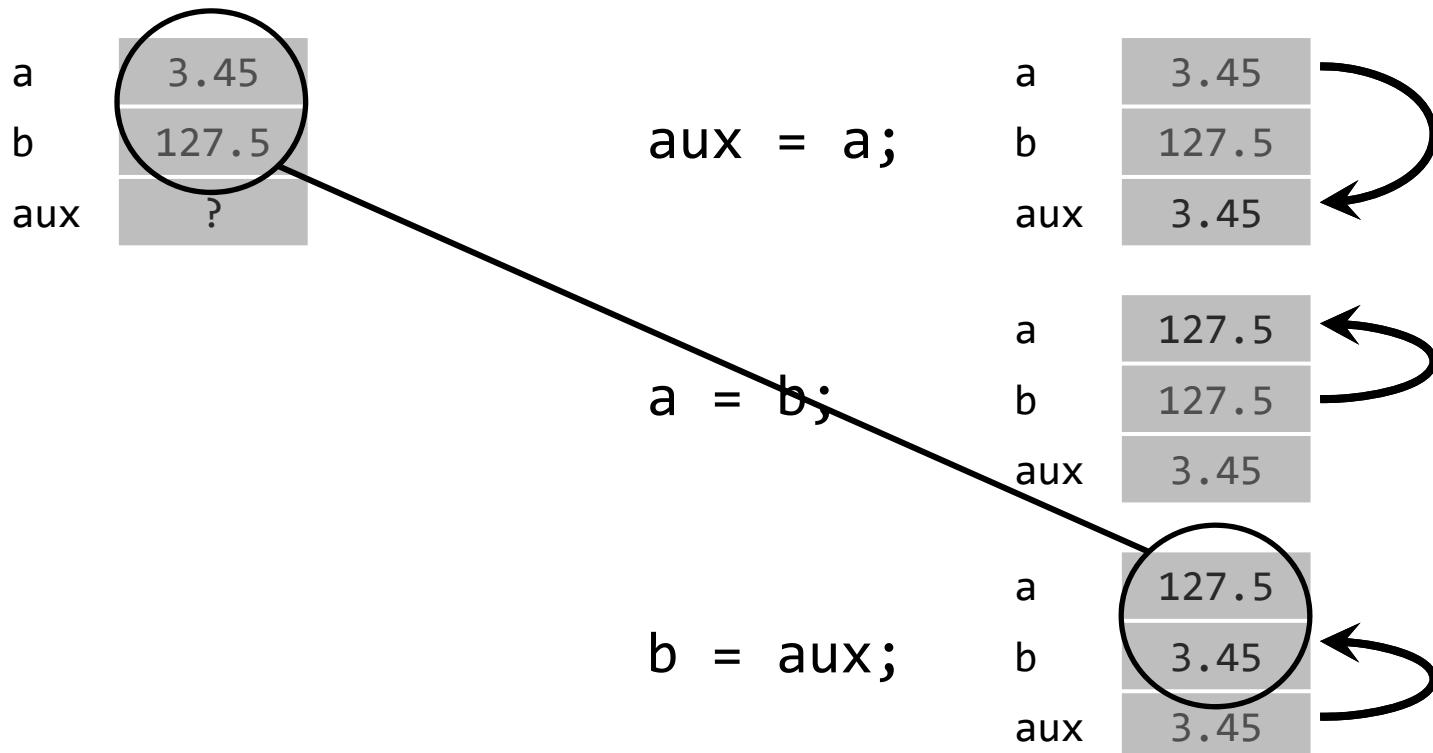


# Ejemplo: Intercambio de valores

Queremos intercambiar el valor de dos variables

- ✓ Necesitamos una variable auxiliar

```
double a = 3.45, b = 127.5, aux;
```



# Variables y memoria

---

Memoria suficiente para su tipo de valores

```
short int i = 3;
```

i 3

```
int j = 9;
```

j 9

```
char c = 'a';
```

c a

```
double x = 1.5;
```

x 1.5

El significado de los bits depende del tipo de la variable:

00000000 00000000 00000000 01111000

Interpretado como **int** es el entero 120

Interpretado como **char** (sólo 01111000) es el carácter 'x'



# Expresiones

---

Una expresión es una secuencia de operandos y operadores:

*operando operador operando operador operando ...*

```
total = cantidad * precio * 1.18;
```

                  |  
                  Expresión



# Variables y expresiones. Ejemplo

---

variables.cpp

```
#include <iostream>
using namespace std;

int main()
{
    int cantidad;
    double precio, total;
    cantidad = 12;
    precio = 39.95;
    total = cantidad * precio;
    cout << cantidad << " x " << precio << " = "
        << total << endl;

    return 0;
}
```



# Ejemplo de uso de variables (I)

---

```
#include <iostream>
using namespace std;

int main()
{
    int cantidad;
    double precio, total;
```

Memoria	
cantidad	?
precio	?
total	?
...	



# Ejemplo de uso de variables (II)

---

```
#include <iostream>
using namespace std;

int main()
{
    int cantidad;
    double precio, total;
    cantidad = 12;
```

Memoria	
cantidad	12
precio	?
total	?
	...



# Ejemplo de uso de variables (III)

---

```
#include <iostream>
using namespace std;

int main()
{
    int cantidad;
    double precio, total;
    cantidad = 12;
    precio = 39.95;
```

Memoria	
cantidad	12
precio	39.95
total	?
	...



# Ejemplo de uso de variables (IV)

```
#include <iostream>
using namespace std;

int main()
{
    int cantidad;
    double precio, total;
    cantidad = 12;
    precio = 39.95;
    total = cantidad * precio;
```

Memoria	
cantidad	12
precio	39.95
total	479.4
	...



# Ejemplo de uso de variables (V)

```
#include <iostream>
using namespace std;

int main()
{
    int cantidad;
    double precio, total;
    cantidad = 12;
    precio = 39.95;
    total = cantidad * precio;
    cout << cantidad << " x " << precio << " = "
        << total << endl;
```

cantidad

12

precio

39.95

total

479.4

...

D:\FP\Tema2>variables  
12 x 39.95 = 479.4



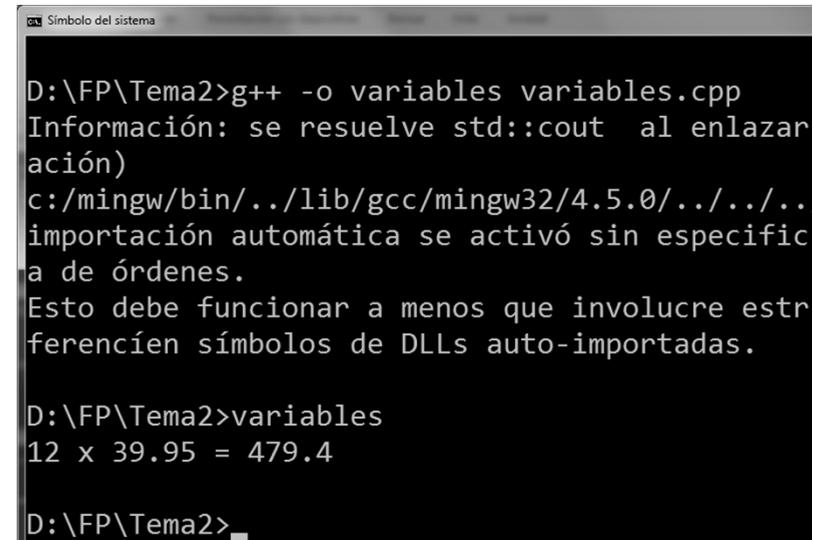
# Ejemplo de uso de variables (VI)

## *Ejemplo de uso de variables*

```
#include <iostream>
using namespace std;

int main()
{
    int cantidad;
    double precio, total;
    cantidad = 12;
    precio = 39.95;
    total = cantidad * precio;
    cout << cantidad << " x " << precio << " = "
        << total << endl;

    return 0;
}
```



```
D:\FP\Tema2>g++ -o variables variables.cpp
Información: se resuelve std::cout al enlazar
ación)
c:/mingw/bin/..../lib/gcc/mingw32/4.5.0/..../..
importación automática se activó sin especific
a de órdenes.
Esto debe funcionar a menos que involucre estr
erencié símbolos de DLLs auto-importadas.

D:\FP\Tema2>variables
12 x 39.95 = 479.4
D:\FP\Tema2>
```



# Conversiones automáticas de tipos

## Promoción de tipos

Dos operandos de tipos distintos: El valor del tipo *menor* se promociona al tipo *mayor*

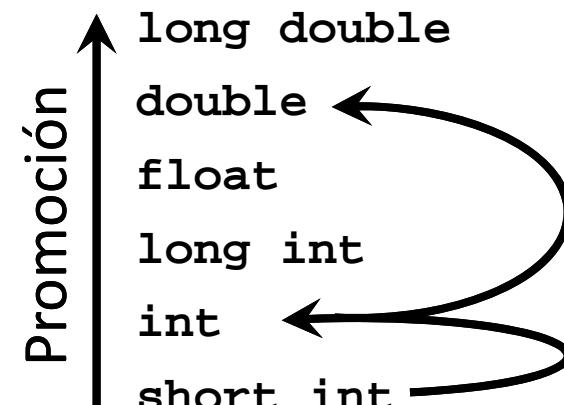
```
short int i = 3;  
int j = 2;  
double a = 1.5, b;  
b = a + i * j;
```

```
b = a + 3 * 2;
```

↳ Valor 3 short int (2 bytes) → int (4 bytes)

```
b = 1.5 + 6;
```

↳ Valor 6 int (4 bytes) → double (8 bytes)



# Conversiones seguras y no seguras

---

✓ Conversión segura: De un tipo menor a un tipo mayor

short int → int → long int → ...

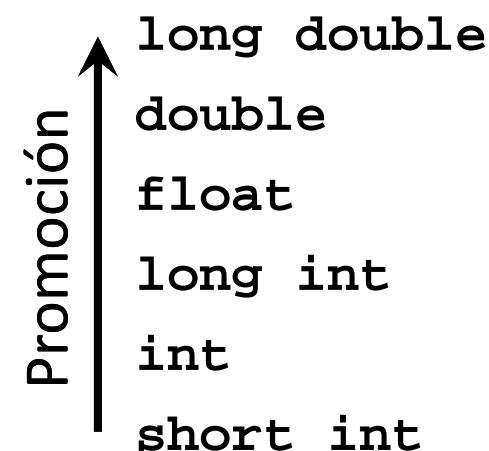
✓ Conversión no segura: De un tipo mayor a un tipo menor

int entero = 1234;

char caracter;

caracter = entero; // Conversión no segura

- Si la variable destino ocupa menos en memoria: Pérdida de información en la conversión



# Moldes (*casts*)

---

Fuerzan una conversión de tipo:

*tipo(expresión)*

El valor resultante de la *expresión* se trata como un valor del *tipo*

```
int a = 3, b = 2;  
cout << a / b;           // Muestra 1 (división entera)  
cout << double(a) / b; // Muestra 1.5 (división real)
```



# Valores proporcionados por el usuario (I)

---

`cin (iostream)`

*character input stream*

Lectura de valores en variables: operador `>>` (*extractor*)

```
cin >> cantidad;
```



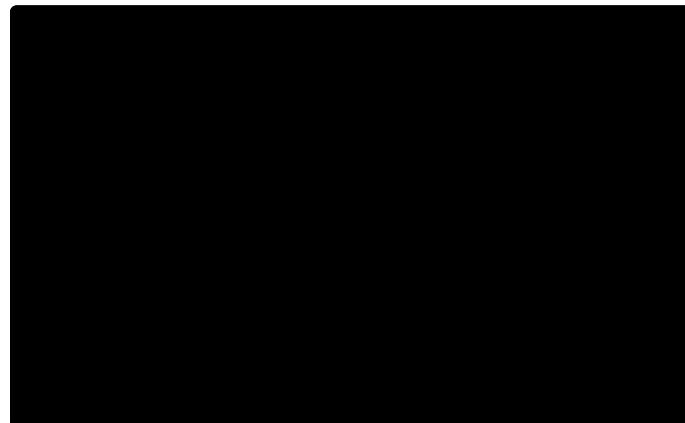
```
cin >> cantidad;
```

Memoria

cantidad 12

...

1 2 ↵



# Valores proporcionados por el usuario (II)

---

*El extractor >>*

`cin >> variable;`

Transforma los caracteres introducidos en datos

Cursor parpadeante: lugar de lectura del siguiente carácter

La entrada termina con Intro (cursor a la siguiente línea)

¡El destino del extractor debe ser SIEMPRE una variable!

Se ignoran los espacios en blanco iniciales



# Valores proporcionados por el usuario (III)

---

Dependiendo del tipo de la variable en la que se va a almacenar el dato leído se lee una u otra cosa:

- **char**: Se lee un carácter en la variable
- **int**: Se leen dígitos y se transforman en el valor a asignar
- **float/double**: Se leen dígitos (quizá el punto y más dígitos) y se asigna el valor
- **bool**: Si se lee 0, se asigna **false**; con cualquier otro valor se asigna **true**



Se amigable con el usuario



Lee cada dato en una línea

```
cout << "Introduce tu edad: ";
cin >> edad;
```



# Lectura de cadenas (string)

```
#include <string>
using namespace std;
```

`cin >> cadena` termina con el primer espacio en blanco

`cin.sync()` descarta la entrada pendiente

```
string nombre, apellidos;
cout << "Nombre: ";
cin >> nombre;
cout << "Apellidos: ";
cin >> apellidos;
cout << "Nombre completo: "
    << nombre << " "
    << apellidos << endl;
```

```
Nombre: Luis Antonio
Apellidos: Nombre completo: Luis Antonio
```

apellidos recibe "Antonio"

```
string nombre, apellidos;
cout << "Nombre: ";
cin >> nombre;
cin.sync(); ←
cout << "Apellidos: ";
cin >> apellidos;
cout << ...
```

```
Nombre: Luis Antonio
Apellidos: Hernández Yáñez
Nombre completo: Luis Hernández
```

*¿Cómo leer varias palabras?  
Siguiente página...*



# Lectura sin saltar espacios en blanco (I)

---

- ✓ Lectura de un carácter sin saltar espacios en blanco:

```
cin.get(c); // Lee el siguiente carácter
```

- ✓ Lectura de cadenas sin saltar los espacios en blanco:

```
getline(cin, cad); → Lee todo lo que haya hasta  
el final de la línea (Intro)
```

- Recuerda: *Espacios en blanco* son espacios, tabuladores, saltos de línea...



# Lectura sin saltar espacios en blanco (II)

---

```
string nombre, apellidos;
cout << "Nombre: ";
getline(cin, nombre);
cout << "Apellidos: ";
getline(cin, apellidos);
cout << "Nombre completo: "
    << nombre << " "
    << apellidos << endl;
```

```
Nombre: Luis Antonio
Apellidos: Hernández Yañez
Nombre completo: Luis Antonio Hernández Yañez
```



# Valores proporcionados por el usuario (IV)

---

## *Lectura de valores enteros (int)*

Se leen dígitos hasta encontrar un carácter que no lo sea

12abc↙    12 abc↙    12    abc↙    12↙

Se asigna el valor 12 a la variable

El resto queda pendiente para la siguiente lectura

## *Lectura de valores reales (double)*

Se leen dígitos, el punto decimal y otros dígitos

39.95.5abc↙    39.95 abc↙    39.95↙

Se asigna el valor 39,95 a la variable; el resto queda pendiente



# Valores proporcionados por el usuario (V)

---

*¿Qué pasa si el usuario se equivoca?*

- ✓ El dato no será correcto
- ✓ Aplicación profesional: código de comprobación y ayuda
- ✓ Aquí supondremos que los usuarios no se equivocan
- ✓ En ocasiones añadiremos comprobaciones sencillas



Para evitar errores, lee cada dato en una instrucción aparte



# Valores proporcionados por el usuario (VI)

*¿Qué pasa si el usuario se equivoca?*

```
int cantidad;  
double precio, total;  
cout << "Introduce la cantidad: ";  
cin >> cantidad;  
cout << "Introduce el precio: ";  
cin >> precio;  
cout << "Cantidad: " << cantidad << endl;  
cout << "Precio: " << precio << endl;
```

*¡Amigable con el usuario!*  
¿Qué tiene que introducir?

```
Introduce la cantidad: abc  
Introduce el precio: Cantidad: 0  
Precio: 1.79174e-307
```

No se puede leer un entero → 0 para cantidad y Error  
La lectura del precio falla: precio no toma valor (*basura*)



# Valores proporcionados por el usuario (VII)

---

*¿Qué pasa si el usuario se equivoca?*

```
Introduce la cantidad: 12abc  
Introduce el precio: Cantidad: 12  
Precio: 0
```

12 para cantidad  
No se puede leer un real  
→ 0 para precio y Error

```
Introduce la cantidad: 12.5abc  
Introduce el precio: Cantidad: 12  
Precio: 0.5
```

12 para cantidad  
.5 → 0,5 para precio  
Lo demás queda pendiente

```
Introduce la cantidad: 12  
Introduce el precio: 39.95  
Cantidad: 12  
Precio: 39.95
```

*¡¡Lectura correcta!!!*



# Programa con lectura de datos (I)

---

## *División de dos números*

*Pedir al usuario dos números y mostrarle el resultado de dividir el primero entre el segundo*

Algoritmo.-

1. Pedir el numerador

Variable numerador (double)

2. Pedir el denominador

Variable denominador (double)

3. Realizar la división, guardando el resultado

Variable resultado (double)

resultado = numerador / denominador

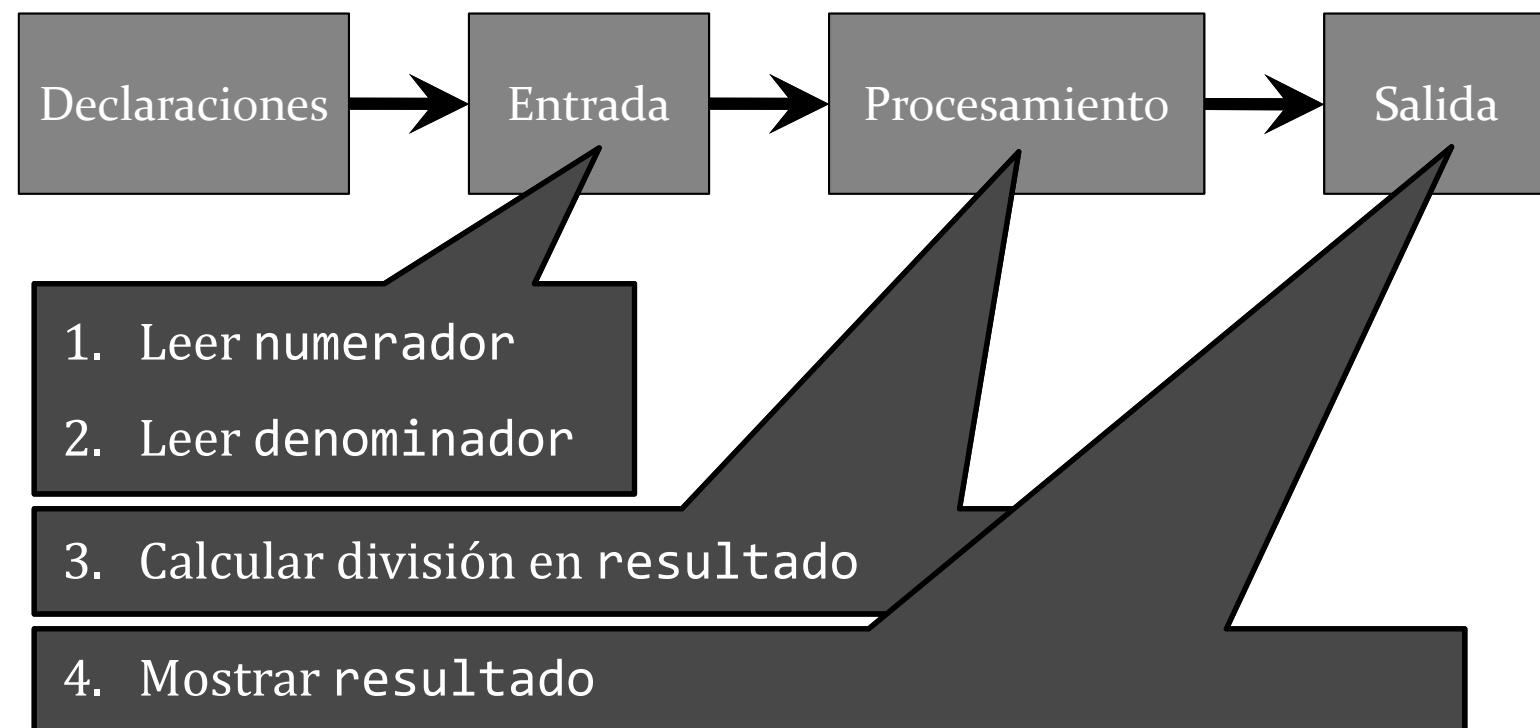
4. Mostrar el resultado



# Un esquema general

## *Entrada-Proceso-Salida*

Muchos programas se ajustan a un sencillo esquema:



# Programa con lectura de datos. Instrucciones

---

## *División de dos números*

*Pedir al usuario dos números y mostrarle el resultado de dividir el primero entre el segundo.*

1. Leer numerador

```
cin >> numerador;
```

2. Leer denominador

```
cin >> denominador;
```

3. Calcular división en resultado

```
resultado = numerador / denominador;
```

4. Mostrar resultado

```
cout << resultado;
```



# Programa con lectura de datos. Implementación

## *División de dos números*

división.cpp

```
#include <iostream>
using namespace std;
```

```
int main()
{
```

Declaraciones    double numerador, denominador, resultado;

Entrada        cout << "Numerador: ";
                  cin >> numerador;
                  cout << "Denominador: ";
                  cin >> denominador;

Procesamiento    resultado = numerador / denominador;

Salida         cout << "Resultado: " << resultado << endl;

```
    return 0;
```

```
}
```

129

Denominador: 2

Resultado: 64.5



# Resolución de problemas. Análisis / Diseño

---

## Problema

*Dadas la base y la altura de un triángulo, mostrar su área*

Refinamiento

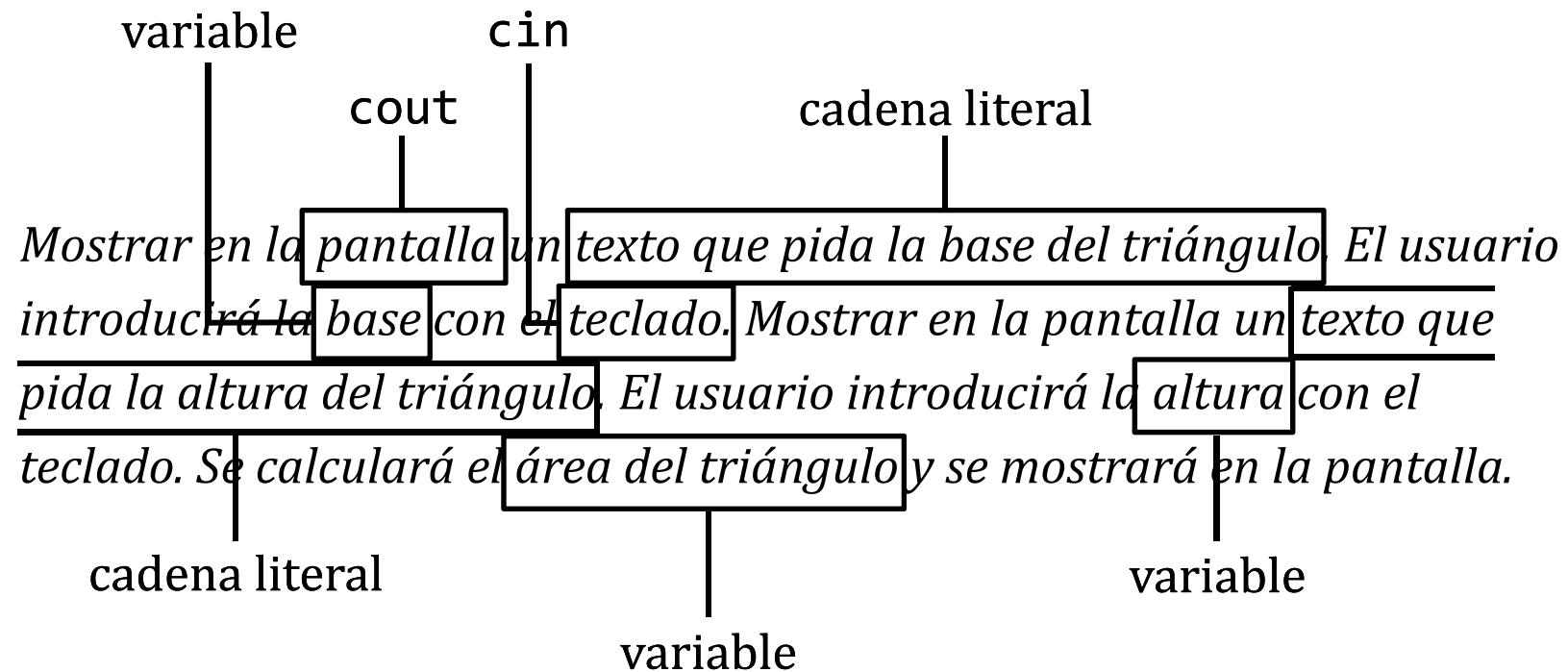
*Mostrar en la pantalla un texto que pida la base del triángulo. El usuario introducirá el valor con el teclado. Mostrar en la pantalla un texto que pida la altura del triángulo. El usuario introducirá el valor con el teclado. Se calculará el área del triángulo y se mostrará en la pantalla.*



# Resolución de problemas. Objetos

---

*Objetos: Datos que maneja el programa*



# Resolución de problemas. Tipos

---

*Datos que maneja el programa: tipos*

<i>Objeto</i>	<i>Tipo</i>	<i>¿Varía?</i>	<i>Nombre</i>
Pantalla		Variable	cout
"Introduzca la base del triángulo: "		Constante	<i>ninguno</i>
Base del triángulo	double	Variable	base
Teclado		Variable	cin
"Introduzca la altura del triángulo: "		Constante	<i>ninguno</i>
Altura del triángulo	double	Variable	altura
Área del triángulo	double	Variable	area



# Resolución de problemas. Operaciones

---

## *Operaciones (acciones)*

`cout << ...`

`cin >> ...`

*Mostrar en la pantalla un texto que pida la base del triángulo. El usuario introducirá la base con el teclado. Mostrar en la pantalla un texto que pida la altura del triángulo. El usuario introducirá la altura con el teclado. Se calculará el área del triángulo y se mostrará en la pantalla.*

`area = base * altura / 2`



# Resolución de problemas. Algoritmo

---

Secuencia de acciones que ha de realizar el programa para conseguir resolver el problema

1. Mostrar en la pantalla el texto que pida la base del triángulo
2. Leer del teclado el valor para la base del triángulo
3. Mostrar en la pantalla el texto que pida la altura
4. Leer del teclado el valor para la altura del triángulo
5. Calcular el área del triángulo
6. Mostrar el área del triángulo



# Resolución de problemas. Programa (I)

```
#include <iostream>
using namespace std;
int main()
{
```

Declaraciones

Algoritmo  
traducido  
a código  
en C++

1. Mostrar en la pantalla el texto que pida la base del triángulo
2. Leer del teclado el valor para la base del triángulo
3. Mostrar en la pantalla el texto que pida la altura del triángulo
4. Leer del teclado el valor para la altura del triángulo
5. Calcular el área del triángulo
6. Mostrar el área del triángulo

```
return 0;
```

```
}
```



# Resolución de problemas. Implementación Programa

*El programa: implementación*

triángulo.cpp

```
#include <iostream>
using namespace std;

int main()
{
    double base, altura, area; // Declaraciones
    cout << "Introduzca la base del triángulo: "; // 1
    cin >> base; // 2
    cout << "Introduzca la altura del triángulo: "; // 3
    cin >> altura; // 4
    area = base * altura / 2; // 5
    cout << "El área de un triángulo de base " << base // 6
        << " y altura " << altura << " es: " << area << endl;

    return 0;
}
```

D:\FP\Tema02>triángulo  
Introduzca la base del triángulo: 34.7  
Introduzca la altura del triángulo: 12  
El área de un triángulo de base 34.7 y altura 12 es: 208.2

¿triángulo?



Para ver bien los acentos, eñes y similares, usa la orden: chcp 1252 en la consola



Recuerda: las instrucciones terminan en ;



# Operadores

---

Cada tipo determina las operaciones posibles

Tipos de datos numéricos (`int`, `float` y `double`):

- Asignación (=)
- Operadores aritméticos
- Operadores relacionales (menor, mayor, igual, ...)

Tipo de datos `bool`:

- Asignación (=)
- Operadores lógicos (Y, O, NO)

Tipos de datos `char` y `string`:

- Asignación (=)
- Operadores relacionales (menor, mayor, igual, ...)



# Operadores aritméticos (I)

---

*Operadores para tipos de datos numéricos*

Operador	Operandos	Posición	int	float / double
-	1 (monario)	Prefijo		Cambio de signo
+	2 (binario)	Infijo		Suma
-	2 (binario)	Infijo		Resta
*	2 (binario)	Infijo		Producto
/	2 (binario)	Infijo	Div. entera	División real
%	2 (binario)	Infijo	Módulo	No aplicable
++	1 (monario)	Prefijo / postfijo		Incremento
--	1 (monario)	Prefijo / postfijo		Decremento



# Operadores aritméticos (II)

---

## *Operadores monarios y operadores binarios*

### ✓ Operadores monarios (*unarios*)

- Cambio de signo (-):

Delante de variable, constante o expresión entre paréntesis

-saldo      -RATIO      -(3 \* a - b)

- Incremento/decremento (sólo variables) (prefijo/postfijo):

++interes      --meses      j++      // 1 más ó 1 menos

### ✓ Operadores binarios

- Operando izquierdo    operador    operando derecho

Operandos: literales, constantes, variables o expresiones

2 + 3      a \* RATIO      -a + b

(a % b) \* (c / d)



# Operadores aritméticos (III)

---

*¿División entera o división real?*

/

Ambos operandos enteros: división entera

```
int i = 23, j = 2;  
cout << i / j; // Muestra 11
```

Algún operando real: división real

```
int i = 23;  
double j = 2;  
cout << i / j; // Muestra 11.5
```



# Operadores aritméticos (IV)

*Módulo (resto de la división entera)*

%

Solo aplicable en la división entera ya que no se obtienen decimales y queda un resto

$$\begin{array}{r} 123 \\ \quad\quad\quad | \quad\quad\quad 5 \\ \hline 24 \\ 123 \% 5 \end{array}$$

Ambos operandos han de ser enteros

```
int i = 123, j = 5;  
cout << i % j; // Muestra 3
```



# Operadores aritméticos (V)

## *Operadores de incremento y decremento*

++/--

Incremento/decremento de la variable numérica en una unidad

Prefijo: Antes de acceder (preincremento y predecremento)

```
int i = 10, j;  
i=i+1;           j = ++i; // Incrementa antes de copiar  
j=i;             cout << i << " - " << j; // Muestra 11 - 11
```

Postfijo: Despues de acceder (postincremento y postdecremento)

```
int i = 10, j;  
j=i;           j = i++; // Copia y después incrementa  
i=i+1;         cout << i << " - " << j; // Muestra 11 - 10
```



No mezcles ++ y -- con otros operadores



# Operadores aritméticos. Ejemplo

```
#include <iostream>
using namespace std;

int main() {
    int entero1 = 15, entero2 = 4;
    double real1 = 15.0, real2 = 4.0;
    cout << "Operaciones entre los números 15 y 4:" << endl;
    cout << "División entera (/): " << entero1 / entero2 << endl;
    cout << "Resto de la división (%): " << entero1 % entero2 << endl;
    cout << "División real (/): " << real1 / real2 << endl;
    cout << "Num = " << real1 << endl;
    real1 = -real1;
    cout << "Cambia de signo (-): " << real1 << endl;
    real1 = -real1;
    cout << "Vuelve a cambiar (-): " << real1 << endl;
    cout << "Se incrementa antes (++ prefijo): " << ++real1 << endl;
    cout << "Se muestra antes de incrementar (posfijo ++): "
        << real1++ << endl;
    cout << "Ya incrementado: " << real1 << endl;
    return 0;
}
```

operadores.cpp



# Orden de evaluación

---

*¿En qué orden se evalúan los operadores?*

$$3 + 5 * 2 / 2 - 1$$

¿De izquierda a derecha?

¿De derecha a izquierda?

¿Unos antes que otros?

Precedencia de los operadores (prioridad):

Se evalúan antes los de mayor precedencia

¿Y si tienen igual prioridad?

De izquierda a derecha

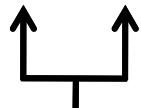
Paréntesis: fuerzan a evaluar su subexpresión



# Precedencia de los operadores

Precedencia	Operadores
Mayor prioridad	cast (molde)
	<code>++ --</code> (postfijos)
	<code>++ --</code> (prefijos)
	<code>-</code> (cambio de signo)
	<code>* / %</code>
Menor prioridad	<code>+ -</code>

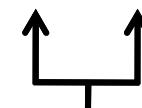
$$3 + 5 * 2 / 2 - 1 \rightarrow 3 + 10 / 2 - 1 \rightarrow 3 + 5 - 1 \rightarrow 8 - 1 \rightarrow 7$$



Misma precedencia:  
Izquierda antes



Mayor  
precedencia



Misma precedencia:  
Izquierda antes



# Evaluación de expresiones

---

$((3 + 5) * 4 + 12) / 4 - (3 * 2 - 1)$  Primero, los paréntesis...

↓ \* antes que -

$(8 * 4 + 12) / 4 - (6 - 1)$

↓ \* antes que + ↓

$(32 + 12) / 4 - 5$

↓

$44 / 4 - 5$

↓ / antes que -

$11 - 5$

↓

6



Pon espacio antes y después  
de cada operador binario



# Una fórmula

fórmula.cpp

Dado un valor x calcula el resultado

```
#include <iostream>
using namespace std;

int main()
{
    double x, f;
    cout << "Introduce el valor de X: ";
    cin >> x;
    f = 3 * x * x / 5 + 6 * x / 7 - 3; ←
    cout << "f(x) = " << f << endl;
    return 0;
}
```

$$f(x) = \frac{3x^2}{5} + \frac{6x}{7} - 3$$



Usa paréntesis para mejorar la legibilidad:

$f = (3 * x * x / 5) + (6 * x / 7) - 3;$



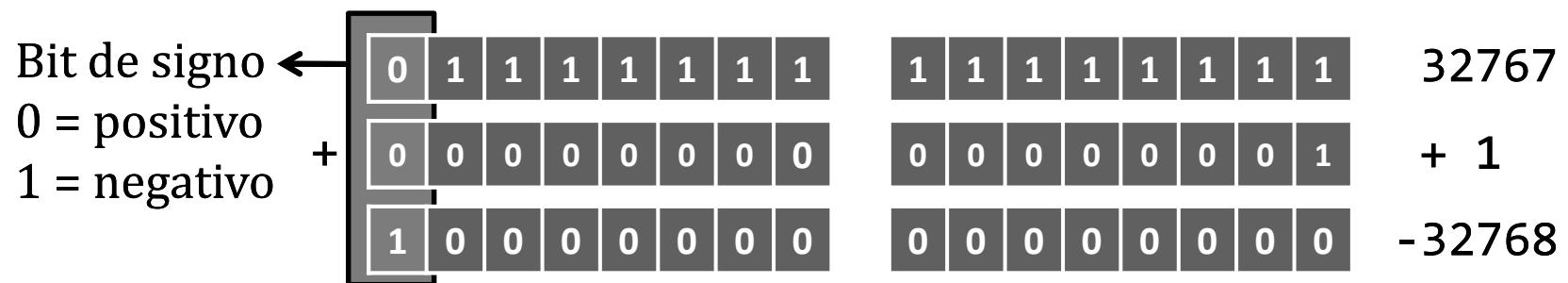
# Desbordamiento

---

*¿Valor siguiente al máximo?*

Valor mayor del máximo (o menor del mínimo) del tipo

```
short int i = 32767; // Valor máximo para short int  
i++; // 32768 no cabe en un short int  
cout << i; // Muestra -32768
```



# Funciones matemáticas

#include <cmath>

---

Algunas ...	abs(x)	Valor absoluto de x
	pow(x, y)	x elevado a y
	sqrt(x)	Raíz cuadrada de x
	ceil(x)	Menor entero que es mayor o igual que x
	floor(x)	Mayor entero que es menor o igual que x
	exp(x)	$e^x$
	log(x)	Ln x (logaritmo natural de x)
	log10(x)	Logaritmo en base 10 de x
	sin(x)	Seno de x
	cos(x)	Coseno de x
	tan(x)	Tangente de x
	round(x)	Redondeo al entero más próximo
	trunc(x)	Pérdida de la parte decimal (entero)



# La biblioteca cmath

mates.cpp

```
#include <iostream>
using namespace std;
#include <cmath> ←
```

$$f(x,y) = 2x^5 + \sqrt{\frac{x^3}{y^2}} - \cos(y)$$

```
int main() {
    double x, y, f;
    cout << "Valor de X: ";
    cin >> x;
    cout << "Valor de Y: ";
    cin >> y;
    f = 2 * pow(x,5) + sqrt( pow(x,3) / pow(y,2) )
        / abs(x * y) - cos(y);
    cout << "f(x,y) = " << f << endl;
    return 0;
}
```

Para usar `pow()` con un número entero:

Usa el molde `double()`:  
`pow(double(i), 5)`



Pon un espacio para separar los paréntesis seguidos



# Operaciones con caracteres

char

---

Asignación, ++/-- y operadores relacionales

*Funciones para caracteres (biblioteca cctype)*

`isalnum(c)` true si c es una letra o un dígito

`isalpha(c)` true si c es una letra

`isdigit(c)` true si c es un dígito

`islower(c)` true si c es una letra minúscula

`isupper(c)` true si c es una letra mayúscula

false en caso contrario

`toupper(c)` devuelve la mayúscula de c

`tolower(c)` devuelve la minúscula de c

...



# Operaciones con caracteres. Ejemplo

```
...
#include <cctype>

int main() {
    char caracter1 = 'A', caracter2 = '1', caracter3 = '&';
    cout << "Carácter 1 (" << caracter1 << ".- " << endl;
    cout << "Alfanumérico? " << isalnum(caracter1) << endl;
    cout << "Alfabético? " << isalpha(caracter1) << endl;
    cout << "Dígito? " << isdigit(caracter1) << endl;
    cout << "Mayúscula? " << isupper(caracter1) << endl;
    caracter1 = tolower(caracter1);
    cout << "En minúscula: " << caracter1 << endl;
    cout << "Carácter 2 (" << caracter2 << ".- " << endl;
    cout << "Alfabético? " << isalpha(caracter2) << endl;
    cout << "Dígito? " << isdigit(caracter2) << endl;
    cout << "Carácter 3 (" << caracter3 << ".- " << endl;
    cout << "Alfanumérico? " << isalnum(caracter3) << endl;
    cout << "Alfabético? " << isalpha(caracter3) << endl;
    cout << "Dígito? " << isdigit(caracter3) << endl;
    return 0;
}
```

caracteres.cpp

$1 \equiv \text{true} / 0 \equiv \text{false}$



# Operadores relacionales (I)

---

## ✓ Comparaciones (*condiciones*)

Condición simple ::= Expresión Operador\_relacional Expresión

## ✓ Concordancia de tipo entre las expresiones

## ✓ Resultado: bool (true o false)

<	menor que
<=	menor o igual que
>	mayor que
>=	mayor o igual que
==	igual que
!=	distinto de

## Operadores (prioridad)

...
* / %
+ -
< <= > >=
== !=
= += -= *= /= %=



# Operadores relacionales (II)

---

Menor prioridad que los operadores aditivos y multiplicativos

```
bool resultado;  
int a = 2, b = 3, c = 4;  
resultado = a < 5;  
resultado = a * b + c >= 12;  
resultado = a * (b + c) >= 12;  
resultado = a != b;  
resultado = a * b > c + 5;  
resultado = a + b == c + 1;
```



No confundas el operador de igualdad (==)  
con el operador de asignación (=)



# Abreviaturas aritméticas

---

*variable* = ~~variable~~ *operador op\_derecho;*  
↑                   ↑  
La misma         ≡  
*variable operador= op\_derecho;*

Asignación	Abreviatura	
a = a + 12;	a += 12;	
a = a * 3;	a *= 3;	Igual precedencia que la asignación
a = a - 5;	a -= 5;	
a = a / 37;	a /= 37;	De momento, mejor evitarlas
a = a % b;	a %= b;	



# Constantes

---

Para indicar que una variable es constante ponemos el modificador de acceso const

Las constantes son variables inicializadas a las que no dejamos variar



```
const short int MESES = 12;  
const double Pi = 3.141592,
```

La constante no podrá volver a aparecer a la izquierda de un =



*Programación con buen estilo:*  
Pon en mayúscula la primera letra  
de una constante o todo su nombre



# ¿Por qué utilizar constantes con nombre?

---

- ✓ Aumentan la legibilidad del código

```
cambioPoblacion = (0.1758 - 0.1257) * poblacion;           VS.  
cambioPoblacion = (RatioNacimientos - RatioMuertes) * poblacion;
```

- ✓ Facilitan la modificación del código

```
double compra1 = bruto1 * 18 / 100;  
double compra2 = bruto2 * 18 / 100;           3 cambios ←  
double total = compra1 + compra2;  
cout << total << " (IVA: " << 18 << "%)" << endl;
```

```
const int IVA = 18;           ¿Cambio del IVA al 21%?  
double compra1 = bruto1 * IVA / 100;  
double compra2 = bruto2 * IVA / 100;           1 cambio ←  
double total = compra1 + compra2;  
cout << total << " (IVA: " << IVA << "%)" << endl;
```



# Constantes. Ejemplo

constantes.cpp

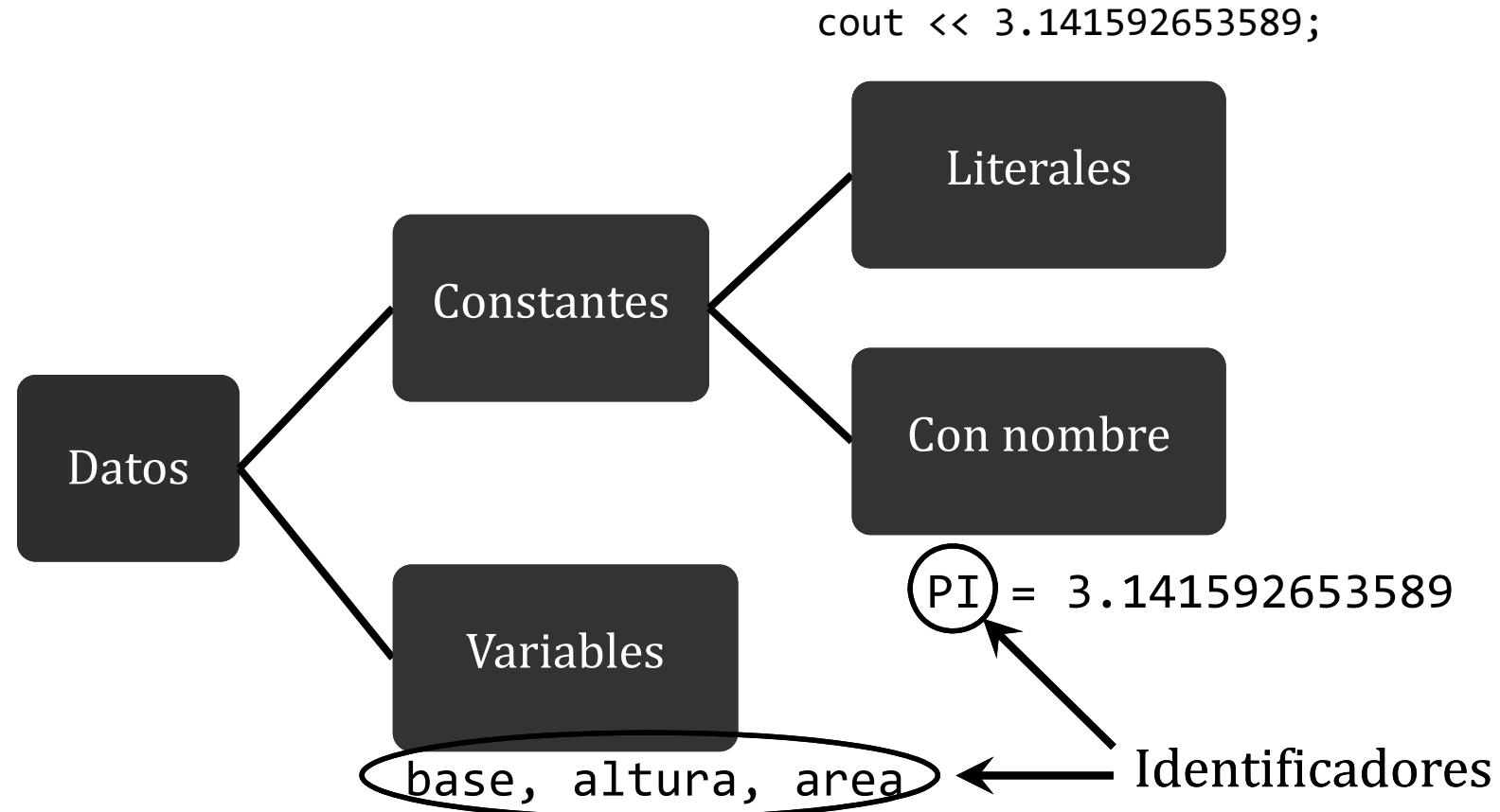
```
#include <iostream>
using namespace std;

int main() {
    const double PI = 3.141592;
    double radio = 12.2, circunferencia;
    circunferencia = 2 * PI * radio;
    cout << "Circunferencia de un círculo de radio "
        << radio << ":" << circunferencia << endl;
    const double EULER = 2.718281828459; // Número e
    cout << "Número e al cuadrado: " << EULER * EULER << endl;
    const int IVA = 21;
    int cantidad = 12;
    double precio = 39.95, neto, porIVA, total;
    neto = cantidad * precio;
    porIVA = neto * IVA / 100;
    total = neto + porIVA;
    cout << "Total compra: " << total << endl;
    return 0;
}
```



# Variabilidad de los datos

---



# Formato de la salida

#include <iomanip>

Constantes y funciones a enviar a cout para ajustar el formato de salida

Biblioteca	Constante/función	Propósito
iostream	showpoint / noshowpoint	Mostrar o no el punto decimal para reales sin decimales (34.0)
	fixed	Notación de punto fijo (reales) (123.5)
	scientific	Notación científica (reales) (1.235E+2)
	boolalpha	Valores bool como true / false
	left / right	Ajustar a la izquierda/derecha (por defecto)
iomanip	setw( <i>anchura</i> )*	Nº de caracteres ( <i>anchura</i> ) para el dato
	setprecision( <i>p</i> )	Precisión: Nº de dígitos (en total) Con fixed o scientific, nº de decimales

\*setw() sólo afecta al siguiente dato que se escriba,  
mientras que los otros afectan a todos



# Formato de la salida

bool fin = false;	0->false
cout << fin << "->" << boolalpha << fin << endl;	
double d = 123.45;	
char c = 'x';	
int i = 62;	
cout << d << c << i << endl;	123.45x62
cout << " " << setw(8) << d << " " << endl;	123.45
cout << " " << left << setw(8) << d << " " << endl;	123.45
cout << " " << setw(4) << c << " " << endl;	x
cout << " " << right << setw(5) << i << " " << endl;	62
double e = 96;	
cout << e << " - " << showpoint << e << endl;	96 - 96.0000
cout << scientific << d << endl;	1.234500e+002
cout << fixed << setprecision(8) << d << endl;	123.45000000



# Acerca de *Creative Commons*

---



## *Licencia CC (Creative Commons)*

Este tipo de licencias ofrecen algunos derechos a terceras personas bajo ciertas condiciones.

Este documento tiene establecidas las siguientes:



Reconocimiento (*Attribution*):

En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.



No comercial (*Non commercial*):

La explotación de la obra queda limitada a usos no comerciales.



Compartir igual (*Share alike*):

La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Pulsa en la imagen de arriba a la derecha para saber más.

